A FRAMEWORK FOR INTELLIGENT NOTIFICATION MANAGEMENT IN MULTITASKING DOMAINS

BY

SHAMSI TAMARA IQBAL

B.C.S., Bangladesh University of Engineering and Technology, 2001
M.S., University of Illinois at Urbana-Champaign, 2004

DISSERTATION

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2008

Urbana, Illinois

Doctoral Committee:

Associate Professor Brian P. Bailey, Chair
Professor Dan Roth
Professor Alex Kirlik
Professor Thomas Huang
Professor Scott Hudson, Carnegie Melon University
Dr. Eric Horvitz, Microsoft Research

Abstract

Interruptions in the workplace are becoming increasingly prevalent due to the proliferation of proactive behavior within communication applications and collaborative practices. Interruptions caused by notifications from communication applications (email, instant messaging clients) or operating systems, phone calls and collocated individuals often cause a forced break in the user's activity as they may require action on the user's behalf or cause them to switch their attention to the incoming request. Research has shown that interruptions at inopportune moments often result in substantial costs to users and their tasks, e.g. frustration and reduced productivity. However, information conveyed by notifications is also often beneficial to users. A current thrust within the HCI community has been to develop solutions that reduce the cost of interruption caused by notifications while maintaining their utility.

In this work, we focus on one class of interruption, *notifications in the desktop*, and present one solution to managing such notifications - *intelligently timing their delivery*. Our solution is based on a deep theoretical understanding of how humans process information and what moments during a user's task execution exhibit lower mental workload. We leverage *breakpoints*, transitions between units of action as potential moments for presenting notifications, as we empirically show these moments to have lower mental workload. Through a series of empirical studies, we demonstrate how presenting interruptions at breakpoints lowers interruption costs and how the cost varies based on the granularity of the breakpoint, how these breakpoints can be detected using known task structures and how breakpoints can be detected without any knowledge of the underlying task. We develop OASIS, a computational framework that can detect and differentiate three levels of breakpoints with reasonable accuracy without requiring any complex machinery. OASIS is the first system of its kind that can detect breakpoints and schedule notifications. We evaluate how OASIS picks breakpoints for novel users and novel tasks, and how scheduling notifications at breakpoints reduces interruption costs. This work provides the first empirical evidence that intelligent notification management benefits the end user and contributes new lessons for designing effective notification management systems.

To my parents

Acknowledgements

I would like to thank my advisor Brian Bailey for his astute guidance and support over the past six years. Without his perseverance, patience and persistence this PhD would not have been possible. His expectation for high standards has been a constant motivation to strive for excellence, and his advising has helped me mature as an independent thinker and creative researcher.

I am also indebted to Karrie Karahalios, Dan Roth, Alex Kirlik, Scott Hudson, Eric Horvitz and Thomas Huang for valuable advice over the course of this PhD. Whenever needed, they have happily provided ideas and suggestions, allowing me to look at problems and solutions from perspectives that may not have been apparent at the onset. Many thanks to Taher Saif, Rumi Shammin and Alok Sutradhar for imparting invaluable lessons from their own experiences regarding Ph.D. completion and job search.

My parents, Kazi M. Iqbal and Nina H. Iqbal, have been a continual source of support and inspiration. Even when I would occasionally dwindle in self-doubt, their belief in me always reinforced my motivation and confidence. I am glad to have finally become the third 'doctor' in the family and fulfill my mom's secret desire, though not the type she had originally envisioned. Their teachings in life have made me what I am today and I am proud to have parents who have always emphasized on following my dreams. My uncles, aunts and cousins have never failed to cheer me on, and their exuberance in my achievements has been particularly touching.

My interactions with my peers have been one of the most rewarding experiences during the past few years. Many a discussion with Jacob Biehl, my 'academic sibling', led to successfully addressing seemingly unsolvable problems, research or otherwise. Over the years we have shared not only a cubicle, but also achievements, successes, occasional frustrations and numerous cups of coffee. Tony Bergstrom and Eric Gilbert have been good cubicle-neighbors and initiators/participants in many inspiring discussions.

My successful completion of the PhD was undoubtedly boosted by a delicate mixture of work and relaxation. I am particularly grateful to have a great group of friends with whom I could completely be myself at the end of a grueling day. Our closely knit group consisting of Rumi Shammin, Tabassum Haque, Alok Sutradhar, Mehruba Firdaus, Tim

Hossain, Tahmina Akhter, Munawar Hafiz, Farhana Ashraf, Sharif Islam, Sabreena Ashraf, Taher Saif, Shahneela Choudhury has been like a big happy family and I fondly recall many escapades over the years that we had undertaken. Special thanks to Fariba Khan, Atanu Khan, Sonia Jahid, Imranul Hoque, Abdullah Muzahid, Abdullah Nayeem, Ahsan Arefin, Yusuf Sarwar, Mehedi Bakht for their continual demonstration in enjoying life and making me a part of it. Special thanks to Professor Salim Rashid and Zeenet Rashid for their roles in taking extra care to make all incoming Bangladeshi students feel at home. Outside of Champaign, Farah Farzana and Ferdous Rubaiyat have been a distant source of support and inspiration over many years and I look forward to many more collocated years ahead in Redmond. I also thank my friends from Viqarunnisa and BUET, among everything else, for asking with unfailing regularity when I plan to finish my Ph.D. Perhaps their enquiries made me more conscious of the fact that I should probably get this over with soon.

Tanya Crenshaw and Erin Chambers have been great ambassadors for the CS department at UIUC and their friendship is particularly valuable to me. They have been a continual source of inspiration and perhaps I have grown more confident in my own abilities because of my association with them. I would also like to thank Ramona Thompson, Nat Thompson, Jodie Boyer, Adam Lee, Jay Patel, William Baker, Mat Yapchaian and Damon Cook for their friendship, suggestions and support over the years.

My best wishes go out to the future of HCI within our department: Tony Bergstrom, Eric Gilbert, Josh Hailpern, Moushumi Sharmin, Brittany Smith, Scarlett Herring among many others. I also wish the very best to the Bangladeshi graduate students in the CS department at UIUC: Munawar Hafiz, Ragib Hasan, Ayesha Yasmeen, Maifi Khan, Fariba Khan, Mehedi Bakht, Abdullah Muzahid, Imranul Hoque, Sonia Jahid, Moushumi Sharmin, Farhana Ashraf, Shameem Ahmed, Yusuf Sarwar, Abdullah Nayeem and Ahsan Arefin. Keep up the tradition!

Special thanks to Mary Beth Kelley, Anda Ohlson and Barb Cicone for taking care of all academic affairs over the years. Whenever there was some administrative issue, I knew that I could count on them to come up with a solution.

Finally, I would like to thank the Almighty for His kind grace that allowed me to successfully complete this Ph.D.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

This dissertation addresses a prevalent and critical problem in human-computer interaction – managing interruptions caused by notifications (McFarlane and Latorella 2002). We use the term notification to refer to visual alerts in the desktop that communicate valuable information to users (Figure 1.1). These may be generated by the operating system and/or applications such as communication clients and calendaring systems. Such notifications provide useful services, including supporting near instant communication (Latorella 1999; Czerwinski, Cutrell et al. 2000; Dabbish and Kraut 2004), maintaining awareness of peripheral information (Maglio and Campbell 2000), reminding upcoming activities (Dey and Abowd 2000), or providing help to users during complex tasks (Maes 1994; Rich and Sidner 1998). An important benefit of notifications is that they convey information to the user automatically, eliminating the need to periodically monitor information sources in anticipation of arrival of new information.

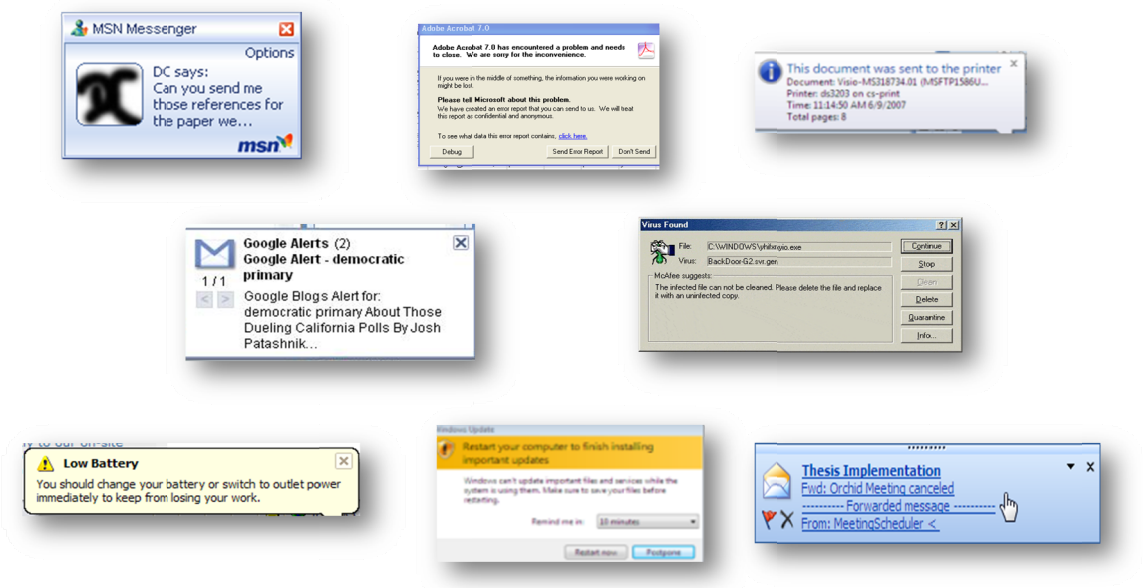


Figure 1.1: Example notifications from the desktop interface.

One significant problem with notifications is they often occur at random moments during task execution (Czerwinski, Horvitz et al. 2004; Mark, Gonzalez et al. 2005; Iqbal and Horvitz 2007). Such interruptions to the ongoing task have been empirically shown to have significant negative impact on users and their tasks. For example, receiving a notification at a random moment causes users to perform tasks slower (Czerwinski, Cutrell et al. 2000; Bailey and Konstan 2006), commit more errors (Kreifeldt and McCarthy 1981; Latorella 1999) and experience increased frustration, annoyance and anxiety (Zijlstra, Roe et al. 1999; Adamczyk and Bailey 2004; Bailey and Konstan 2006).

Such disruptive effects have been observed for a single notification in controlled settings, but the problem becomes even more intensified when considering multiple notifications interrupting users on a daily basis in real environments. Research has shown that for an average knowledge worker, 57.1% of working spheres are interrupted (Mark, Gonzalez et al. 2005), 27% of task switches are due to phone call, emails and reminders (Czerwinski, Horvitz et al. 2004), and 15-20% of an employee's effort are spent on dealing with interruptions (Solingen, Berghout et al. 1998). A recent study has shown that knowledge workers receive around 7 notifications per hour and can spend up to 25 minutes on responding to notifications, subsequent diversions to other tasks and resumption of interrupted activity (Iqbal and Horvitz 2007).

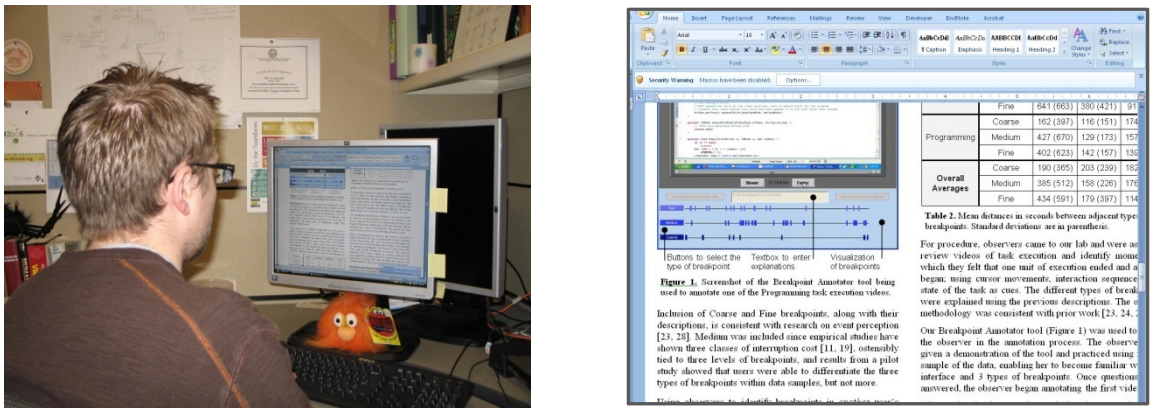


Figure 1.2: Example scenario of a notification interrupting an ongoing task. The user on the left is engaged in a cognitively demanding task. An email notification arrives causing the user to suspend the task and switch to the email client. After reading the email, the user may perform other tasks, taking advantage of the break from his primary task. On return to the primary task, it is often difficult to resume the exact task state as the user was away from the task for a while.

The problem of interruption is not only limited to the desktop, but also is prevalent in other multitasking domains such as aviation cockpits (Dismukes, Young et al. 1998; Latorella 1999), in-vehicle systems (Lee, Hoffman et al. 2004) and control rooms (Stanton 1994). Consequences of an inopportune interruption in these domains can be severe - a small delay in response time or an error in task performance due to even a single interruption could have catastrophic outcomes, e.g. failure to execute live-saving instructions in flight decks or incorporate critical information in decision making in command and control systems (Barnes and Monan 1990; Latorella 1999; McFarlane and Latorella 2002). The common challenge in all these domains is better managing interruptions so that the disruption is mitigated while ensuring that the benefits are maintained.

In this dissertation we describe the design, development and evaluation of OASIS, a framework for intelligently managing notifications in the desktop interface. OASIS is the result of a truly multidisciplinary approach – applying theories and principles from cognitive psychology, statistical modeling and software engineering. OASIS leverages understanding from cognitive psychology to determine lower cost moments for interruption, applies statistical modeling techniques to develop models of these moments, and uses software engineering principles to design and develop a computational framework to intelligently manage notifications using these models. Empirical studies have shown significant benefits of managing notifications using OASIS.

OASIS acts as an intermediary between the user and the application that has information for him or her. When an application attempts to request user attention through a notification, OASIS defers it until moments that have lower interruption costs. One key innovation of the system is in what moments it picks for notification deferral and how. The moments that OASIS leverages are *breakpoints,* a concept borrowed from cognitive psychology. Breakpoints are moments of transition between consecutive units of action. For example, breakpoints may exist between tasks, e.g. between programming and checking email. They may also exist at a finer level *within* the same task, e.g. between compiling and editing code during programming. OASIS monitors interaction events in

real time and uses predefined models of breakpoints to determine breakpoints at multiple levels of granularity.

At a very basic level, the value of OASIS is that it allows users to finish their ongoing thought before switching their focus to an interruption. However, its impact is much more profound. OASIS is one of the first notification management systems that can schedule notifications at lower cost moments and through OASIS we provide the first evidence of benefits of notification scheduling in authentic, complex task settings. OASIS also provides a test bed for different interruption management techniques and policies to be explored in practical settings and opens up new directions of research in this domain.

1.1 Scope

Generally speaking, an interruption is a forced break in the cognitive focus on ongoing activities (Speier, Valacich et al. 1999). Interruption can be initiated by the user herself due to an internal shift in goal (Kahneman 1973; Norman and Shallice 1986; Pashler and Johnston 1998). Interruptions also occur externally through components of the environment in which the user exists, e.g. a physically collocated individual, communication devices such as cell phones or pagers, or desktop notifications such an email or IM alert (Hudson, Christensen et al. 2002; Czerwinski, Horvitz et al. 2004; Iqbal and Horvitz 2007). Such external interruptions are prevalent in the workplace, particularly within the desktop interface, as well as in control rooms, in-vehicle systems and health care environments.

This dissertation addresses interruption in the desktop, focusing on better managing notification delivery. The goal is to reduce the disruptive effects of notifications while ensuring that their benefits are maintained. In this domain, frequency of notifications, urgency and relevance, presentation style and timing of notifications are all important factors that need to be considered in techniques for managing notifications. The approach our work investigates is *notification timing*. We focus on timing as we strongly believe that timing is a fundamental component in mitigating disruption and other factors are often dependant on timing, e.g. urgency and relevance. We develop techniques in controlled setting for determining moments during task execution sequences so that interruption costs are reduced, operationalize these techniques within a computational

system and show effectiveness of the system in reducing interruption costs in authentic task settings.

1.2 Existing Approaches

Although interruption has been an enduring problem within the research community, the focus has been mostly on understanding the magnitude of the effects of the problem in a wide variety of multitasking domains. Developing computational solutions to address the problem has only recently become a focus of the community.

The simplest and most intuitive solution perhaps would be to turn off notifications altogether. This naïve approach could be potentially useful in situations where the information being conveyed is of low priority and users know exactly when information would be available and where to access it. However, it is often difficult to keep track of all potential information sources, running a risk of missing important information and subsequent negative consequences.

A more pragmatic approach is making the information readily available for the user, but allowing the user to decide when they want to access it. This approach is based on the *negotiated* strategy for interruption management proposed by McFarlane and Latorella (2002). In this setting, information is usually made available on a peripheral visual display and users negotiate their own interruptible moments (Van Dantzich, Robbins et al. 2002; McCrickard, Catrambone et al. 2003; Stasko, Miller et al. 2004) . While this approach works well for information that is less frequent or less urgent, for the more common case of arrival of frequent and/or important information this may cause substantial increases in self-interruptions.

An alternate solution is to have a third party or a system decide on behalf of the user when is a good moment to interrupt. This falls into the *mediated* approach category in McFarlane's parlance. The challenge in this approach is for a system to automatically determine interruptible moments during a user's task execution, taking into account the user's ongoing task state as well as the context of the notification.

Two threads of research have emerged in this domain. The first focuses on investigating effects of interrupting at different stages during task execution in controlled laboratory settings (Czerwinski, Cutrell et al. 2000; Adamczyk and Bailey 2004; Bailey and Konstan

2006). The goal is to develop techniques for managing notifications that can be eventually adapted to authentic settings. These techniques, however, have not been tested in these types of authentic settings, primarily due to lack of automated systems that implement them. The other thread focuses on developing models of interruptibility, mapping interaction events and environmental sensors to discrete levels of interruptibility (Horvitz, Breese et al. 1998; Horvitz, Jacobs et al. 1999; Horvitz and Apacible 2003; Horvitz, Kadie et al. 2003; Hudson, Fogarty et al. 2003; Fogarty, Hudson et al. 2004; Horvitz, Koch et al. 2004; Fogarty, Ko et al. 2005). It has not been empirically demonstrated if delivering notifications at moments that a system identifies as interruptible using these models does indeed reduce interruption costs, primarily due to the lack of operational systems. This dissertation attempts to fill in this existing void in the interruption literature, by contributing a notification management system that delivers notifications at moments that have been shown to have lower interruption costs, and providing empirical results demonstrating the benefits of using the system on users and their tasks.

1.3 Our Solution

Our solution addresses the notification management problem by developing a system, OASIS, which delivers notifications at moments during task execution that have lower interruption costs. OASIS implements a new technique for deferring notifications until *breakpoints* during task execution, where breakpoints are defined as moments of transition between units of tasks at multiple levels of granularity (Newtson 1973; Newtson and Engquist 1976; Zacks and Tversky 2001). The motivation behind this technique comes from earlier work that showed that breakpoints have lower interruption costs (Adamczyk and Bailey 2004; Bailey and Konstan 2006) compared to interruptions at random moments during tasks. This in turn was motivated by Miyata and Norman's theoretical postulations that breakpoints indicate moments of lower workload and thus may be more opportune for interruption (Miyata and Norman 1986).

OASIS acts as an intermediary between the user and an application wanting to interrupt him or her. An application wanting to convey some information to the user sends a request to OASIS. OASIS monitors user interactions and allows the notification to go

through when a breakpoint has occurred. Using the system to deliver notifications at breakpoints during complex tasks, we showed both qualitative and quantitative benefits of our approach (Iqbal and Bailey 2008).

One of the key technical innovations of our work is demonstrating how the perceptual structure of a task can be leveraged to identify different levels of breakpoints without any underlying knowledge of the task (Iqbal and Bailey 2007; Iqbal and Bailey 2008). OASIS supports three levels of breakpoints as potential interruptible moments – *Coarse*, as moments of transitions between *largest* meaningful task units; *Fine*, as moments of transition between *smallest* meaningful task units; and *Medium*, which exists between the two. Our work has shown that these three types of breakpoints can be reliably detected within a user's task execution sequence by only monitoring the desktop event stream.

OASIS does not directly consider benefits of the notification – rather assumes availability of a time window within which the notification will retain its value for the user. If the time window expires, the notification is no longer valuable to the user; therefore the goal is to ensure that the notification is delivered within that time window. We also make the assumption that the notification will be equally valuable to the user throughout the duration of the time window, meaning that the value will not diminish with time. The underlying reasoning algorithm takes into account the time window while detecting breakpoints for notification scheduling. Mapping benefits of the notification on to a time window is a separate research problem and is beyond the scope of this dissertation.

The value of using OASIS is that it provides a range of flexible defer-to-breakpoint policies that can be leveraged to schedule notifications at moments that are known to have lower interruption costs. With notifications having varying levels of relevance and importance to the user, these policies provide opportunities for better balancing cost and benefits of notifications. An evaluation of the system showed effectiveness of the notification deferral approach in reducing interruption costs as compared to random delivery of notifications.

Our work contributes a system for scheduling notifications at moments that are known to have lower underlying costs, pushing techniques that have been explored in controlled setting into the field. We have demonstrated how to overcome the technical challenges in

designing and developing such a system. We have also provided first evidence from the field showing that the system is effective. Overall, lessons from the preliminary studies, implementation and evaluation all contribute towards a deeper understanding of how notifications should be managed using the mediated approach in practice, opening up many new directions for future research.

1.4 Illustrative Example

We explain how our system works through an illustrative example. In this example, the user is performing a programming task, which requires him to occasionally check for references and also make notes for documentation. The user's manager occasionally checks upon his progress through MSN messenger. The user's work primarily straddles Visual Studio for coding, Firefox for checking references as well as unrelated browsing, and Microsoft Word for documentation. OASIS is installed on the user's machine so that it can schedule notifications from Outlook and MSN messenger at suitable breakpoints. All three applications that the user is interacting with are instrumented to send application levels events to OASIS, as well as hooks within the windows event stream that send system level events.

An email about an upcoming vacation arrives in his Outlook inbox, triggering Outlook to send a request to OASIS. This email needs to be delivered to the user within the next thirty minutes. OASIS adds the request to a queue and checks the current deferral policy for notifications, which is *defer-until-Coarse*. By monitoring application and system level events and evaluating them against statistical models of breakpoints, it waits for the next Coarse breakpoint to occur.

Fifteen minutes later, the user saves his code, minimizes his Visual Studio window and switches to ebay.com to check on some recent bids. OASIS notes this switch in activity and identifies it to be a Coarse breakpoint. Since a notification is queued that is supposed to be delivered at a Coarse breakpoint, OASIS lets it go through to the user at this moment.

Later, the user returns to his code and refocuses on the programming task. After a while, an instant messaging notification request from the user's manager, is generated and sent to OASIS. The deferral policy for this notification is *defer-until-medium*. OASIS, as before, adds this request to the queue and waits for a Medium breakpoint to occur.

Five minutes later, the user finishes compiling his code and switches to Firefox to open MSDN. Based on the corresponding events, OASIS identifies this to be a Medium breakpoint, which matches the stated deferral policy. OASIS then grants the notification request and the user attends to it.

The example above illustrates how the user is allowed to maintain an acceptable level of awareness while delivering the information at a moment that is known to have lower interruption costs. Since the information was deferred slightly, until a suitable breakpoint, OASIS allowed the user to focus on their task while experiencing less negative affect. Through this process, users were allowed to finish their ongoing tasks and then react to the notification, rather than being interrupted while in the middle of coding. The value of this work is that it allows these types of example scenarios to be realized in practice. In later chapters we will describe the empirical benefits of this approach.

1.5 Contributions

This dissertation makes the following contributions:

i) *Results from user studies showing that mental processing effort is temporarily reduced at breakpoints, resulting in lower interruption costs.*

   The underlying approach in this work is leveraging breakpoints during task execution as opportune moments for interruption. Miyata and Norman postulated that breakpoints are more opportune for interruption since they indicate moments of lower mental workload. However, they never empirically tested these claims. Our work contributes empirical evidence supporting these theoretical postulations linking a physiological measure of workload to mental processing efforts (Iqbal and Bailey 2004; Iqbal, Adamczyk et al. 2005; Bailey and Iqbal 2008). We show that users experience transitory reductions in workload at breakpoints. We also provide the novel discovery that the reduction is not uniform – breakpoints between coarser perceptual units have larger reductions than breakpoints between finer units. Through a follow up study, we showed empirical results that show that interrupting at breakpoints with lower workload results in lower interruption costs (Iqbal and Bailey 2005). These findings provide scientific explanation as to why breakpoints are more

opportune for interruption and emphasize the need to differentiate breakpoints based on their predicted interruption costs. This will be discussed in detail in Chapter 4.

ii) *Demonstration of how task structure can be used to differentiate breakpoints based on predicted interruption costs.*

Prior work has shown that breakpoints vary based on mental workload and that mental workload affects interruption costs. However, measuring workload is extremely difficult in uncontrolled settings and therefore is not feasible for most desktop tasks. Our work demonstrates how the structure of a task can be used to differentiate breakpoints based on predicted interruption costs without explicitly having to measure workload (Iqbal and Bailey 2006). Our work leverages characteristics of the task structure to effectively predict three classes of interruption cost and show generalizability of the approach across a set of desktop tasks with predefined task structure. These details are described in Chapter 5.

iii) *A new technique for identifying breakpoints without knowledge of the task.*

Our work provides a new and innovative technique for identifying breakpoints (Iqbal and Bailey 2007). Prior work required detailed descriptions of tasks to detect where breakpoints occurred within those tasks (Bailey, Adamczyk et al. 2006). The key innovation of this work is that it can identify breakpoints without any knowledge of the underlying task. This is accomplished through leveraging only events within the desktop event stream and application level events. Three levels of breakpoints are detected and differentiated through this process. This technique is not only useful for notification management systems that reason about scheduling notifications at breakpoints, but also can be beneficial in identifying switches in task groups in within task organization applications (Dragunov, Dietterich et al. 2005), or to understand cognitive organization of complex tasks, such as programming. These details are provided in Chapter 6.

iv) *A fully functional framework for managing notifications.*

This work contributes a fully functional framework for managing notifications, leveraging understanding of cognitive processing in determining lower cost moments

for interruption (Iqbal and Bailey 2008). This is one of the first systems of its kind that allows techniques developed in the lab to be realized within authentic settings. The framework detects breakpoints as lower cost moments by implementing previously developed techniques for breakpoint detection and differentiation, and interfaces with proactive applications in scheduling notifications at breakpoints. The framework supports a range of flexible defer-to-breakpoint policies to provide different balances between costs of interruption and timeliness of information delivery. The framework also provides a test bed for investigating various interruption management techniques developed in laboratory settings and policies in context of real user tasks in complex settings. We provide details of the system in Chapter 7.

v) *Results and lessons demonstrating the effectiveness of the mediated approach of notification management in realistic settings.*

Several prior studies have shown benefits of scheduling notifications at breakpoints in controlled settings. Our work provides the first evidence showing how using a fully automated system for breakpoint identification and notification scheduling affects interruption costs during real tasks in real settings (Iqbal and Bailey 2008). Our findings show empirical benefits of this approach, and highlight some of the complexities in notification management that arise from deploying the system in practical settings. We provide new lessons for notification management systems and provide directions for further research in this space. We present these details in Chapter 8.

CHAPTER 2

Related Work

Interruption in the workplace has been an important area of focus for HCI researchers over the years. Research in this domain includes understanding effects of interruption, solutions to mitigate disruption caused by interruption and evaluation of solution approaches. Our work focuses on developing and evaluating solutions for more effective management of notifications that often interrupt users with valuable information. The underlying goal is to allow users to continue receiving the notifications that they desire, while being less disrupted by their arrival. More precisely, our work investigates manipulating the timing of notifications, a promising and emerging strategy for interruption management.

In this chapter we discuss existing work in five key areas that are most relevant to our work and situate our work within the existing body of literature. First, we provide an overview of the types of interruptions that occur in the workplace and clarify the types of interruptions that this work addresses. Second, we justify our use of *breakpoints* as potential opportune moments for interruption leveraging existing theories and research. Third, we discuss related approaches in notification timing and also touch upon other factors that impact disruption caused by notifications. Fourth, we discuss how systems are deploying these strategies in practice to better balance between information conveyance and mitigation of the disruption caused by interruption. Finally, we discuss evaluations of systems that reason about when to interrupt and how our work presents further evidence demonstrating the positive impact of scheduling notifications on users and their tasks.

2.1 Prevalence of Interruptions

Interruption is an inevitable part of multitasking users' work routine (O'Conaill and Frohlich 1995; Hudson, Christensen et al. 2002; Czerwinski, Horvitz et al. 2004; Mark, Gonzalez et al. 2005; Iqbal and Horvitz 2007; Iqbal and Horvitz 2007). Such

interruptions can be broadly classified into two categories – external and internal. External interruptions are initiated by an outside agent, beyond control of the user , e.g. phone calls (Green 2000), interruption from a coworker (Hudson, Christensen et al. 2002; Hudson, Fogarty et al. 2003), and application-initiated interruptions, e.g. email notifications (Jackson, Dawson et al. 2001), instant messaging (Cutrell, Czerwinski et al. 2001), software agents (Maes 1994) and system alerts. Internal interruptions, on the other hand, are initiated by the user herself. Self-interrupting the current task to switch to another by choice is habitual for users who are constantly multitasking (Kahneman 1973; Pashler and Johnston 1998; Wickens 2002) and often a result of willed and deliberate conscious control of behavior (Norman and Shallice 1986). Such internal interruptions are occasionally welcomed by users to consciously move away the current task and refocus with new vigor on return from the interruption (DeMarco and Lister 1999).

Interruptions are closely weaved into routine work practices of technology users, as shown in many contemporary studies. Mark et al. (Mark, Gonzalez et al. 2005) showed in a field study with 24 users that 57.1% of the working sphere segments were interrupted, where a working sphere is defined by clusters of events related and oriented towards a common purpose. These interruptions were both external, from coworkers; and internal, where the user herself interrupted the ongoing task to switch to another. Managers were found to experience more external interruptions (59.2%) than internal interruptions (40.8%); where developers and analysts experienced external and internal interruptions about equally. 77% of the interrupted tasks were resumed on the same day.

Phone calls were found to account for 14% of task switches in a study conducted by Czerwinski, Horvitz et al. (2004) whereas email, meetings and appoints reminders accounted for a further 13%. In their study, though, self interruptions (40%) constituted a substantial amount of task switches. Jackson, Dawson et al. (2001) showed how emails pervade our daily life. Solingen, Berghout et al. (1998) show that 15-20 percent of an employee's effort is spent on dealing with interruptions, and 15-20 minutes are spent on addressing each interruption. DeMarco and Lister (1999) show that developers routinely receive around 15 phone calls a day, causing them to routinely suspend ongoing activities.

Chong and Siino (2006) recognized interruptions to be a necessary and inevitable feature of collaborative work. Analyzing 242 interruptions from about 40 hours of field data, they identified factors such as interruption content, length, type and management strategies to impact effects of interruption for programmers working in pairs, compared to individual programmers. Programmers working in pairs dealt with external interruptions by having one member participate in the dialogue with the interrupter while the other would remain focused on the ongoing task. This indicates that knowledge workers are aware of both the benefits and disadvantages of such interruptions and develop their own strategies to effectively compensate for them.

Studies from the field strongly demonstrate that interruptions are a natural and inevitable part of normal work practice. While untimely interruptions may often disrupt the ongoing work flow, the benefits of notifications, emails, phone calls or face-to-face communications cannot be disregarded. Rather than attempting to eliminate the source of interruption, a more practical solution is finding a balance between the cost and benefits of interruptions.

In this dissertation, we focus on providing better support for managing one type of interruption –application initiated interruptions from email, IM notifications or other proactive applications. Though this work focuses on the desktop domain, we believe that principles and lessons from this research can be extended and applied for applications in other domains, e.g. safety critical domains and to address environmental interruption.

2.1.1 Benefits of Interruption

Interruptions from proactive systems typically entail conveyance of information, which often offer substantial benefits to the user. These benefits include supporting near instant communication (Latorella 1996; Czerwinski, Cutrell et al. 2000; Dabbish and Kraut 2004), maintaining awareness of peripheral information (Maglio and Campbell 2000), reminding upcoming activities (Dey and Abowd 2000) or helping users perform complex tasks (Maes 1994; Rich and Sidner 1998).

In terms of maintaining communications in the desktop domain, Czerwinski and Horvitz (2002) have identified benefits of interruptions from instant messaging applications, particularly in collaborative settings. These include knowledge about availability of personal contacts, near instantaneous and rapid, asynchronous

communications and the ability to carry on several informal communications simultaneously. Similarly, Dabbish and Kraut (2004) show that in a team setting where team members are collaborating to achieve common goals, users are more amenable towards interruptions, regarding them as beneficial to the collaborative task and are more willing to adopt interrupting strategies to minimize disruption to their collaborators, e.g. use an awareness display showing current workload of the collaborator.

In terms of task performance and management, Maglio and Campbell (2000) and Maes (1994) note the importance of maintaining awareness of peripheral information such as news headlines, status of background tasks, and assistance on application usage, information filtering and information retrieval etc. Conveyance of peripheral information ostensibly results in interruption to the primary ongoing task, but the effects of the interruption can be mitigated through careful manipulation of the timing or the display modalities of the information. Similarly, reminders of upcoming activities are considered an important facet of task management for many multitasking users, but these reminders also routinely interrupt ongoing tasks. Dey and Abowd (2000) investigate how the contextual information can be leveraged through technology support to better schedule these reminders to ensure timeliness while minimizing disruption to the ongoing task. These and other work strongly demonstrate the importance of interruptions in multitasking situations, as they ensure that information is delivered to the user saving users the effort to seek information themselves.

2.1.2 Costs of Interruption

Studies have empirically demonstrated that due to the lack of interruption presentation policies, benefits of interruptions are often overshadowed by high costs resulting from interruptions at inopportune moments. Inopportune interruptions can negatively impact task completion time (Kreifeldt and McCarthy 1981; McFarlane 1999; Czerwinski, Cutrell et al. 2000; Czerwinski, Cutrell et al. 2000; Cutrell, Czerwinski et al. 2001; Monk, Boehm-Davis et al. 2002), error rate (Latorella 1998) , decision making (Speier, Valacich et al. 1999) and affective state such as frustration, anxiety and annoyance (Zijlstra, Roe et al. 1999; Adamczyk and Bailey 2004; Bailey and Konstan 2005). Bailey and Konstan (2005) manipulated timing of presenting a peripheral task to desktop users engaged in primary tasks and measured cost of interruption along both performance and affective

dimensions. Their results showed that presenting interruptions at random moments during primary tasks resulted in up to 30% increase in task completion time, twice the errors and up to twice the increase in negative affect as compared to interrupting at task breakpoints. Similarly, Speier, Valacich et al. (1999) demonstrated that users experiencing interruptions performed complex decision tasks less accurately and required more time to complete the task and that accuracy decreased and performance time increased with the increase in the frequency of interruptions. Kreifeldt and McCarthy (1981) found that interruptions caused users to perform slower on calculator-based tasks due to the time needed to reorient to a previously suspended task after an interruption. Gillie and Broadbent (1989) found users to perform slower on interrupted tasks and suggest that that the disruptive effects depend on at least the cognitive load required by the interrupting task and its similarity to the primary task.

Iqbal and Horvitz (2007) have shown in a separate study that on average 10 minutes are spent on 'chains of diversions' caused by interruption and an additional 15 minutes were spent on attempts to resume suspended task activities. Their findings showed that 27% of suspended tasks are never recovered during the session. DeMarco reported that the average recovery time after an interruption is 15 minutes (DeMarco and Lister 1999), while Gonzalez and Mark (2004) found the time spent on recovery to be around 25 minutes, similar to the findings of Iqbal and Horvitz.

Despite the purported costs, knowledge workers still depend on notifications to deliver information that is generated in the background of their attention. A focus of interruption researchers therefore has been to develop techniques to better manage notifications by balancing between the costs and benefits. This dissertation contributes to this domain through developing, implementing and evaluating one such technique – deferring notifications until breakpoints during user tasks where this balance may be better achieved.

2.2 Breakpoints

This dissertation leverages the use of breakpoints as its main deferral strategy. A breakpoint represents the moment of transition between two observable, meaningful units of task execution (Newtson 1973), and reflects transitions in perception or action (Zacks

and Tversky 2001). The idea of mitigating effects of interruption by linking cognitive processing to breakpoints was first proposed by Miyata and Norman (1986). They postulated that periods of low workload in a user's task sequence were more opportune (lower cost) for interruption and that these periods of low workload exist at breakpoints between subtasks, i.e. after the evaluation of a subtask and before planning of the subsequent subtask. However, they did not provide empirical evidence supporting their claims.

In this section we present the concept of breakpoints and situate them in the context of resource theories of attention, to further explain why breakpoints may have lower interruption costs. We then describe techniques for breakpoint identification for physical tasks as motivation for the techniques we use in this dissertation to identify breakpoints during interactive tasks.

2.2.1 Concept of a Breakpoint and Its Theoretical Grounding

Several psychology studies have shown the existence of breakpoints to indicate salient moments within task execution that can be used for task unitization (Newtson 1973; Newtson and Engquist 1976; Zacks, Tversky et al. 2001). Newtson and Engquist (1976) demonstrate that observers perceptually organize observed behavior into chunks of meaningful actions at multiple levels of granularity, and that behavioral units are formed at breakpoints. They postulate that users are more aware when one perceptual unit ends and the next one begins. In contrast, while *within* an activity, users are more immersed in performing the action and have a higher level of cognitive processing. This has implications for interruption management. If users are caused to switch tasks at these moments, they can resume the suspended activity more easily since the recognition memory for breakpoints are superior to the recognition for non-breakpoint moments.

Hanson and Hirst (1989) set out to explore how orientation and organization of behavioral units affect how information in a behavioral episode is encoded by an observer. Their findings showed that as people perceive events, they presumably build a hierarchical representation of the underlying action. The depth of this hierarchy depends on the coarseness of the perceived events. The notion of a *breakpoint hierarchy* was further explored by Zacks, Tversky et al. (2001). In their work, they revisited the notion of event perception to understand how people use the temporal hierarchical structure of

events to perceive, plan, understand and act, and most importantly, how people segment activities into events as they happen. In their description, events can be viewed at various levels of abstraction and the depth in the hierarchical structure determines the granularity of the abstraction level. In a related study, the same authors show that outside observers are able to identify coarser breakpoints between the 'largest meaningful units', and finer breakpoints between the 'smallest meaningful units', and that the finer breakpoints results from hierarchical decomposition of the units separated by the coarser breakpoints (Zacks, Tversky et al. 2001). They view fine breakpoints to occur when simplest physical changes occur within orders of seconds. The next level of breakpoints occurs when some intentional act ends, usually within 10-30 seconds. The authors observe that as the time scale increases, events become less physically characterized and more defined by the goals, plans, intentions and traits of their participants. This observation is in line with the concept of GOMS where goals are decomposed hierarchically into sub-goals and finally unit actions that help achieve those sub-goals (Card, Moran et al. 1983).

This indicates that there is an inherent structure in user behavior and this structure can be perceived by independent observers. More importantly, the perceived structure is mostly similar across observers, indicating that there must be common features that observers are leveraging to identify these structures. Since a subset of the fine breakpoints typically align with the coarse breakpoints, mental schemas driving perception and action are thought to have at least a two-level hierarchy. This implies that at least two types of breakpoints should be identifiable – Coarse and Fine – during task execution.

Although prior work investigating the role of breakpoints in perception of task structure, the question remains unanswered as to why breakpoints would have lower interruption costs. One explanation may be provided through cognitive resource theories. Miyata and Norman (1986) postulated that processing efforts are reduced at subtask breakpoints (boundaries) in a hierarchical task decomposition – when users release resources from the previous subtask and are about to start acquiring resources for the upcoming subtask. However, they did not provide empirical evidence supporting this claim and also did not provide insight on which breakpoints (if any) are have lower cost

of interruption among the many subtask breakpoints that exist in a hierarchical decomposition of a task.

Building on Zack's theoretical work and applying Miyata and Norman's theoretical postulations about breakpoints having lower interruption cost, Adamczyk and Bailey (2004) showed that interrupting between coarse grained activities caused less disruption as compared to interrupting between fine grained activities, indicating that different breakpoints at different levels of hierarchy may have different costs of interruption.

Based on this prior work, part of this dissertation focuses on identifying and comparing different breakpoints that exist in a hierarchical decomposition of any task as potential moments for interruption. Our work differs from prior work in that i) it investigates in further details the relationship between workload, task structure and effects of interruption and ii) it elucidates the connection between theory and application by applying the understanding in an operational system.

2.2.2 Identification of Breakpoints

As breakpoints demarcate transitions between cognitively discrete units, it appears as though these points have distinctive properties that differentiate them from other moments during task execution. Newtson and colleagues conducted a series of studies showing this postulation to hold for breakpoints during physical tasks (Newtson and Engquist 1976). Their work tested two potential bases for how breakpoints serve to discriminate action units. One is that actions are defined by achievement of distinct states by the actor of the task that are meaningful. Breakpoints, then, could consist of actor positions that define the occurrence of a recognizable goal state. The other interpretation is that breakpoints are distinguished by a distinctive change having occurred, rather than a distinctive state having been achieved. Results from their studies showed that breakpoints are typically tied to patterns of changes in behavior streams. The authors conclude that if actions are perceptually defined at breakpoints, then the set of breakpoints should contain the *perceptual* structure of an ongoing behavior stream.

The theory behind and the process of identifying *where* such breakpoints occur in behavioral streams has been primarily based on Heider's unit attribution process (Heider 1958), where a continuous stream of information from one person's behavior is organized into meaningful units by independent observers of that behavior. The underlying

assumption is that if many independent observers agree upon the distribution of the units, then this common understanding can be used to define the perceptual structure of that behavior. This idea of encoding behavioral streams using independent observers has been since adopted by many psychology researchers to perceive and segment behaviors. Newtson and colleagues (1973) used a similar unit marking procedure where observers viewed sequences of actions on 16 mm black and white film, and marked breakpoints using a button connected to a computer which recorded the timestamp of the button push. Any marks within a 1 sec-interval were considered the same, and a marked moment was selected to be an 'agreed upon breakpoint' for intervals with marks at least one standard deviation above the mean marks across the entire sequence. Results showed this approach to successfully measure the perceptual unit of ongoing behavior. Later, Hanson and Hirst (1989) used the same procedure to identify breakpoints where the cutoff for a marked interval to be considered as a widely agreed upon breakpoint was set to be 1.65 standard deviations above the mean. The reason why this process works is likely because breakpoints may contain information that sets them apart from other moments during the task. Zack and Tversky cite Barker and Wright's work (Barker and Wright 1954), to identify six characteristics associated with event states that can be indicative of breakpoints at different levels. These are: change in the sphere of behavior between verbal, social and intellectual, change in a predominant part of the body, change in the direction of the behavior, change in the object of the behavior, change in the behavior setting, and a change in the tempo of the activity.

Based on prior studies identifying breakpoints for physical tasks, it appears that there are some commonalities in breakpoints that are not seen during other moments during task execution, which independent observers can detect. One of the goals of this dissertation is to identify whether analogous features exist for breakpoints during interactive tasks, whether these features can be used to distinguish several levels of breakpoints, and whether these features can be recognized by an automated notification management system that reasons about when to interrupt.

2.3 Importance of Notification Timing and Other Factors

The notion of manipulating the timing of a notification has its roots in the taxonomy in managing interruption proposed by McFarlane and Latorella (2002). They present methods to coordinate interruptions that could be integrated into a multitasking work stream to maintain balance between awareness of peripheral information and mitigating their disruptive effects. They discuss four basic coordination solutions: *immediate, scheduled, negotiated* and *mediated*. Our methodology falls into the *mediated* category.

To better manage awareness of information while minimizing disruption, the *mediated* strategy allows a third party/mediator system to decide *when* to interrupt the user. One of the five main approaches for this strategy as proposed by McFarlane is predicting when the user will be most interruptible. Fundamental to this approach is understanding when to time a notification so that the cost and the benefits of the notification are balanced.

Inopportune timing of an interruption with relation to the primary task execution has been found to be one of the more important factors behind its disruptive effects. Based on cognitive breakdown of task execution as discussed by Miyata and Norman (1986), Czerwinski, Cutrell et al. (2000) performed a series of experiments investigating the effects of interrupting primary tasks during their planning, execution and evaluation phases. In their experiments, users were found to take more time to switch to the interrupting task when it was presented during the execution phase of a primary task, indicating that users felt a greater urge to rehearse their position in a task when interrupted during the execution phase as compared to other phases. They also found that interruptions disrupt the evaluation phase more than planning or execution phases in terms of task performance time. However, they did not investigate effects of interrupting at the breakpoints between phases, rather, they focused on interrupting during the different phases themselves. Monk et al. conducted similar studies (Monk 2004; Monk, Boehm-Davis et al. 2004), investigating differences in attentional switching costs for interruptions occurring at various task stages. They measured attentional switching costs in terms of resumption lag, the time required to resume the primary task after attending to the interrupting task. Their results showed that interrupting at a boundary, such as just before beginning a task had significantly less resumption lag than interrupting during a subtask. However, they only tested points before beginning a task as breakpoints,

whereas there are numerous breakpoints in a hierarchical task breakdown and effects of interruption at those points are not well researched. Latorella (1996) explored the effect of interrupting between versus within procedures in a simulated flight deck environment. Results showed that interruptions between tasks caused less reaction time than interruptions within tasks. This finding is further supported and theoretically explained through our preliminary studies in this dissertation, described in Chapter 4.

All work mentioned so far demonstrates the importance of considering timing in notification management. However, apart from timing, other characteristics of an interruption could potentially be managed to mitigate the disruptive effects. These include frequency of an interruption (Cohen 1978; Cohen 1980; Speier, Valacich et al. 1999; Zijlstra, Roe et al. 1999; Monk, Boehm-Davis et al. 2004), relevance and similarity to the ongoing primary task (Gillie and Broadbent 1989; Czerwinski, Cutrell et al. 2000), or presentation of the information conveyed by the interruption (Oberg and Notkin 1992; Maglio, Barrett et al. 2000; Maglio and Campbell 2000; Plaue, Miller et al. 2004; Gluck, Bunt et al. 2007). Presentation schemes for a notification in particular, are closely tied to the negotiated strategy of interruption management, where users "negotiate" when they want to attend to an interruption. The information being conveyed by the interruption is presented in a peripheral or ambient display such as in (Maglio, Barrett et al. 2000; Van Dantzich, Robbins et al. 2002; Plaue, Miller et al. 2004) and it is the user's responsibility to monitor the arrival of information.

While we recognize opportunities for further reducing interruption costs through manipulating various characteristics of interruptions, this dissertation focuses on one aspect – the timing of an interruption. The underlying approach is to find an opportune moment for presenting the notification to the user, within a given time window where the benefit of the notification will be maintained. Our work is unique compared to other work in this domain in the sense that it leverages theories of attention and cognitive processing to determine moments that are opportune for interruption, i.e. breakpoints.

2.4 Systems that Reason About Interruption

In (Horvitz, Jacobs et al. 1999; Horvitz and Apacible 2003; Horvitz, Kadie et al. 2003; Hudson, Fogarty et al. 2003; Fogarty, Hudson et al. 2004; Bailey, Adamczyk et al. 2005;

Fogarty, Ko et al. 2005), researchers are developing computational systems that reason about when to interrupt. The general approach is to balance the benefit of the interruption with the cost of interrupting the primary task (Horvitz, Jacobs et al. 1999) and reason about when to present the interruption during the user's task execution. The major threads of research in this domain are: developing models for predicting interruption costs, operationalizing this within a computational system to detect moments with low interruption costs and scheduling notification delivery at those moments.

2.4.1 Developing Models for Predicting Cost of Interruption

Developing models of interruptibility to predict costs of interruption has been an important thread of research towards realizing interruption management in practice. These models typically leverage cues related to desktop activity, visual and acoustical analysis of the physical environment, and scheduled user actions (Horvitz, Breese et al. 1998; Horvitz and Apacible 2003; Horvitz, Kadie et al. 2003; Hudson, Fogarty et al. 2003). For example, Horvitz and Apacible use these cues to infer a probability distribution over users' attentional state, from which interruption costs are computed (Horvitz and Apacible 2003). The attentional states were classified into low, medium and high cost using retrospective labeling where users went through a video of their interactions. In (Horvitz, Jacobs et al. 1999), the authors developed Bayesian models using information about a user's activity and location that can be used to predict probabilities of a user's attentional focus. Some of the features included scheduled appointments, time of day, proximity of deadlines, acoustics in the users' office to indicate ongoing conversation, status and configuration of software applications etc. The authors also develop a theoretical cost model for computing utility of a message as a determinant of whether it should be delivered to the user at the current moment. The expected utility of relaying information to the user is computed as the difference between the cost of delaying the notification and the cost of interrupting the user with the information.

Horvitz and colleagues in (Horvitz, Breese et al. 1998) use Bayesian modeling techniques to learn patterns of when users require assistance while performing spreadsheet tasks. In a Wizard of Oz setting, experts observed users activity patterns and

attempted to interpret user needs and goals. The predictive features that were found to be predictive of users needing assistance, for example, were *difficulty of current task,* and *recent menu surfing* among others. Likewise, user distraction was found to be influenced by task difficulty and users pausing after an activity. The authors also take into account the important issue of temporal reasoning about predictive events in their models. Using markov models, they model the dependencies among goals in adjacent time slices. A prediction for any given moment ideally will take into account all preceding time slices, weighted with a time decay factor.

In Bayesphone (Horvitz, Koch et al. 2005), Horvitz and colleagues develop models predicting the utility of interrupting users at meetings through phone calls. They gather information from users' meeting calendars about the location, duration, intiator etc and collect annotations from users about the potential cost value (high, medium and low) of interrupting those meetings. Based on the annotations and the values of the features, Bayesian networks are created providing the probability of the cost of interrupting given an instance of a meeting. They also compute the prior probability of attending a meeting, which contributes to the *expected* cost. So the expected cost is a function of the cost of interrupting the meeting given the user is attending and the cost of interrupting the meeting if the user is not attending. The expected value of the information (EVI) is computed to reduce uncertainty about the user attending a meeting. Contrary to other models for predicting interruptibility, this model did not use real time data for prediction and therefore could not predict a continuous cost of interruption. Model accuracies were 81%. Similar models for predicting meeting attendance had 92% accuracy.

In a related project geared towards supporting collaboration and communication, Horvitz and his colleagues develop Coordinate, a prototype service for forecasting presence and availability of computer users. Coordinate differs from earlier systems in that it not only provides estimates of *current* state of user presence, but also provides *future* forecasts of when the user may be available. Coordinate gathers information from multiple devices in the proximity of the user, in addition to information collected from online task management tools such as to-do-lists and schedulers.

Fogarty et al. built statistical models that map interaction events (typing, scrolling, browsing, etc.) to one of three classes of task engagement (Fogarty, Ko et al. 2005),

where reaction time to a secondary task was used as ground truth. This measure of interruptibility was based on performance, as opposed to the retrospective labeling used in prior work. The authors used Expectation Maximization algorithms to cluster the reaction time values into three distinct clusters – interruptible, engaged and deeply engaged. Activity features related to programming tasks were then analyzed to map predictive features to these three levels of interruption cost using a Naïve Bayes classifier. The models, however, were found to adequately distinguish two levels of cost – interruptible and non-interruptible. With 23 predictive features, the models reported 71.8% accuracy, compared to the 58.5% base accuracy of predicting all moments as interruptible. In prior work in this thread of research, the authors collected self interruptibility reports from a group of managers, researchers and interns, and associated environmental sensors with five levels of interruptibility. However, their findings showed that a binary model of interruptible and non-interruptible provided the highest accuracy. This indicated that environmental sensors may be only appropriate for differentiating interruptible moments from non-interruptible moments, and more intricate measures would be needed for further distinction. Among other features, factors indicating social engagement was found to be most predictive for the interruptibility models for managers, resulting in a model accuracy of 87.7%. For researchers, conversation in the last minutes was most predictive and model accuracy was 81%. Finally, for interns, long conversations were one of the more salient predictive features and model accuracy was 80.1%.

While models in this corpus of work have been shown to effectively predict interruptibility, they have either not been utilized for scheduling notifications or the effects of scheduling notifications have not been studied in authentic settings. Our work demonstrates how these models can be realized within a computational system to detect breakpoints during tasks in authentic settings. This allows notifications can be delivered at breakpoints detected in real time and provides opportunity for studying effects of scheduling notifications at breakpoints.

2.4.2 Scheduling Notifications at Interruptible Moments

In the limited number of notification management systems that exist, some form of decision making is required to decide when to interrupt. In the *Priorities* project (Horvitz,

Jacobs et al. 1999) and in the *Notification Platform* (Horvitz, Kadie et al. 2003), the authors use manually constructed Bayesian networks to infer a user's attention and an automated criticality classifier for emails, and relay email notification to the user when the net value of the notification (criticality – cost) is positive. Hudson, Fogarty et al. (2003) empirically showed that certain environmental events indicate whether the user is interruptible or not on a continuum (scale of 1 to 5), but decision making about interruption is more accurate while deciding when *not* to interrupt versus any other level of interruptibility. Fogarty, Ko et al. (2005) similarly showed that certain desktop events can be used to classify users into two classes – interruptible and deeply engaged and suggest that systems should interrupt only during the interruptible states. However, other work showed that user workload changes lies on a continuum and suggests that the cost of interruption may be reflected by similar distributions (Iqbal, Adamczyk et al. 2005). Nonetheless, no evidence has been provided showing this to be indeed the case.

Related systems have focused on supporting management of interruption starting from interruption arrival, suspension of ongoing task to attend to the interruption and recovery and resumption of suspended tasks (Daniels 2000; Franke, Daniels et al. 2002). Franke et al. discuss a mixed initiative spoken dialogue based client-server system which prioritizes interruptions based on user tasks and delivers the information at the end of the current task based on its importance. However, their system has been implemented only within strictly prescribed task e.g. managing requests for supplies using military protocols and no real evaluation of the system has been published.

Only a few systems exist that schedule, or are capable of scheduling notifications. For example, the Lookout system predicts a user's dwell time on a communication message based on an analysis of its content (Horvitz 1999). This prediction is then used to schedule delivery of automated assistance. This system uses a support vector machine (SVM) analysis on the message text to infer probabilities of user goals or tasks within the context of interacting with email. The Notification Platform modulates flow of messages from multiple sources to devices by performing ongoing decision analysis (Horvitz, Kadie et al. 2003). In this system, messages are delivered using the device and modality that is most beneficial for the user. The user's attentional focus is predicted using a dynamic Bayesian network, where the probability of the user being in a particular

attentional state is determined based on temporal dependencies over a variety of environmental and activity related sensors. The predicted attentional state is then used to compute an expected cost of interruption, where the utility of being interrupted is also taken into account. A related system, BESTCOM, considers social and task context, available channels, and communication preferences to select the best timing and modality for interpersonal communications (Horvitz, Koch et al. 2002). Other similar systems have also been developed (Begole, Matsakis et al. 2004; Fogarty, Lai et al. 2004). However, none of these systems have been evaluated to measure the impact of notification scheduling on users and their tasks.

Relative to this work, a major contribution of our research is that we studied the impact of scheduling notifications on users and their tasks using an automated system. We studied one technique for scheduling, deferring notifications until breakpoints. However, our results may help understand the effects of scheduling notifications using other similar systems and help improve their design.

2.5 Evaluation of Systems That Reason About Interruption

Systems that reason about interruption are typically evaluated from the perspective of system performance, based on how well they recognize a particular context or how well they perform actions that are postulated to reduce negative impacts of interruption(Horvitz, Koch et al. 2002; Horvitz and Apacible 2003; Begole, Matsakis et al. 2004). For example, the Priorities system was evaluated in terms of how well it could perform a variety of actions based on the criticality of an email message (Horvitz, Jacobs et al. 1999). In Lilsys (Begole, Matsakis et al. 2004), the authors investigated how well the system communicates the state of the user by collecting qualitative data on the hardware interface elements, user image control and so on. While tuning the system so that its performance is maximized is important, without measuring impact on user experience the value of the system is not fully understood.

Other evaluation include testing the accuracy of models of interruption, as described in (Horvitz and Apacible 2003; Fogarty, Hudson et al. 2004; Fogarty, Ko et al. 2005). While most models are based on environmental cues that are purported to indicate interruptibility, Fogarty et al. (Fogarty, Ko et al. 2005) based their model on reaction

time, a direct measure of interruptibility. Their evaluation, is one of the few evaluations that explicitly investigates effects of using their models on user experience.

In our work we evaluate our system both from the perspective of system performance as well as from the perspective of user experience. We believe that to deploy the system in real life settings, both system performance and impact on user experience need to be evaluated. We investigate the system performance by determining how well the system identifies breakpoints using composite models of breakpoints created from the data of several people. We measure impact of different decision algorithms on the user experience to determine the simplest yet efficient algorithm for scheduling interruption. Involving the user in the evaluation process in addition to evaluating the system is consistent with good HCI practice and is of utmost importance in developing a system that has real life benefits, apart from serving as a research tool.

In summary, our research contributes a novel system for managing notifications. The system leverages approximations of workload at subtask breakpoints in a model for computing cost of interruption based on resource theories of attention. These models are used within a system to schedule interruptions at low cost moments. Integration of workload theory to determine opportune moments for interruption into an interruption reasoning system separates our work from existing systems, since this directly integrates consideration of human cognition and perception while scheduling notifications to interrupt users. Our work complements existing systems which are more focused on environmental cues while our work is focused on task activity and indirectly, the user's internal state. Furthermore, our evaluation of the system looks at both system performance and impact on user experience. Finally, our work provides a common framework for studying alternative algorithms for interruption management, which to our best knowledge, is the first instance of any such framework.

Chapter 3

Thesis Overview

In this chapter we provide an overview of the work in this dissertation, detailed descriptions will be provided in subsequent chapters. We begin this dissertation with a theoretical exploration into what moments are more opportune for interruption, taking into account theories of human cognition and perception (Chapter 4). We focus on breakpoints, moments of transition between cognitive or perceptual systems, and proceed to empirically demonstrate why they may be opportune for interruption. This theoretical framing provides the necessary grounding of breakpoints as an established class of interruptible moments, and allows for development of generalizable policies across a wide variety of tasks. We provide insights on how these theoretical findings can be realized within practical methods and techniques for detecting moments for interruption.

In Chapter 5, we investigate how models of interruption costs at breakpoints can be developed. We focus on predicting interruption costs at breakpoints for tasks with known execution sequences, and leverage structure of the task for identifying predictive features, based on our theoretical findings. Development of models to predict interruption costs provides the first step towards realizing interruption management techniques shown to work in controlled settings for real life authentic tasks, using a fully automated interruption management system.

In Chapter 6, we move on to investigating techniques for identifying breakpoints during free-form interactive tasks without having any knowledge of the underlying task. This allows us to develop methods that an interruption management framework can use for a wide variety of desktop tasks. This is necessary in order to ensure widespread acceptability of these types of systems for managing notifications.

In Chapter 7, we present Oasis, our notification management framework. We describe the design and architecture of the individual components, how they interact to identify breakpoints and schedule notifications at those breakpoints, and describe the different

defer-to-breakpoint policies that the framework supports. This system provides a mechanism to investigate the important question - Do the techniques developed in labs using Wizard of Oz have similar positive effects in the real world, when the process of identifying the interruptible moments and scheduling notifications is fully automated? The system also provides opportunities to explore other models of interruptibility and notification deferral policies, benefiting the research community as a whole.

In Chapter 8, we describe the evaluation of the performance of the framework, and effects of using the framework to schedule notifications during authentic tasks in field settings. This is the first instance of any evaluation of interruption management policies in the field, where identification of interruptible moments and scheduling of notifications is fully automated. Lessons from the study provide valuable insights into how well these techniques work and provide directions for next steps.

In Chapter 9, we provide an overview of the work and situate it within existing work in the domain of interruption management. We revisit lessons learned and outline the next steps of this research to realize benefits of this work in full for realistic, authentic tasks.

We provide a brief summary and conclude this dissertation in Chapter 10.

CHAPTER 4

Empirically Understanding How Workload Changes at Breakpoints during Interactive Tasks

As discussed in Chapter 1, breakpoints represent moments of transition between two observable, meaningful units of task execution, and reflect internal transitions in perception or action. Empirical studies show that scheduling notifications to occur at certain breakpoints can mitigate interruption cost (Czerwinski, Cutrell et al. 2000; Czerwinski, Cutrell et al. 2000; Cutrell, Czerwinski et al. 2001; Adamczyk and Bailey 2004; Bailey and Konstan 2006), compared to other moments in a user's task execution sequence. Miyata and Norman's influential work (Miyata and Norman 1986) provides some explanation as to why this is the case. They theorize that notifications would be less disruptive if they interrupted users at lower workload moments. They further speculated that these moments occur at breakpoints during goal directed tasks. However, the veracity of this assumption has never been empirically tested, particularly in the domain of interactive tasks.

In addition, interactive tasks can be decomposed into recursive patterns of goal formulation and execution, creating many breakpoints at many levels within a task's hierarchical structure (Card, Moran et al. 1983). It is thus unclear as to *which* of these breakpoints would have lower processing demands. This further highlights the need to better understand how workload changes across different levels of breakpoints within the structure of goal-directed tasks.

In this chapter, we describe a study investigating how workload changes during interactive task execution, focusing on breakpoints. To formulate methods used in the study for analyzing workload corresponding to interactive tasks, a preliminary study was

conducted. The preliminary study explored the use and applicability of pupil dilation as a measure of workload for interactive tasks. In a subsequent study we tested how well workload at breakpoints correspond to costs of interruption. The results provide empirical evidence providing an explanation as to why interrupting at breakpoints results in lower cost. We conclude with a discussion on the implications of our findings.

4.1 Exploring Use of Pupil Dilation as a Measure of Workload in Interactive Computing Environments

In order to understand workload changes during task execution, the first step was to determine an appropriate measure that can be used to approximate mental workload in interactive computing environments. After a review of the literature, combined with local availability of needed equipment, we chose pupil dilation as our measure of workload for this work. Using pupil dilation as a measure of mental workload offers the following advantages (Kramer 1991). For example, this measure is *continuous* meaning that it provides a steady stream of workload data; it measures allocation of attentional resources in a *holistic* manner rather than specific pools; it has *low latency*, usually responding to a change in workload in 300-500ms; and it is *immediate*, a few recent data samples indicate workload, which simplifies analysis of the data. However, careful experimental control must be maintained with pupil dilation, as it can be considerably affected by environmental factors such as changes in ambient illumination or screen luminance.

Although pupil size has been shown to correlate well with workload induced by stimulus-driven tasks (Hess and Polt 1964; Juris and Velden 1977; Beatty 1982; Hoecks and Levelt 1993; Hytintk, Tommola et al. 1995), it is not known whether this correlation holds in interactive environments and for tasks with more complex structures than those previously studied. Thus, in the first step of our research we wanted to test whether pupil dilation would correlate with the workload of interactive tasks. Our goal was to answer the following questions:

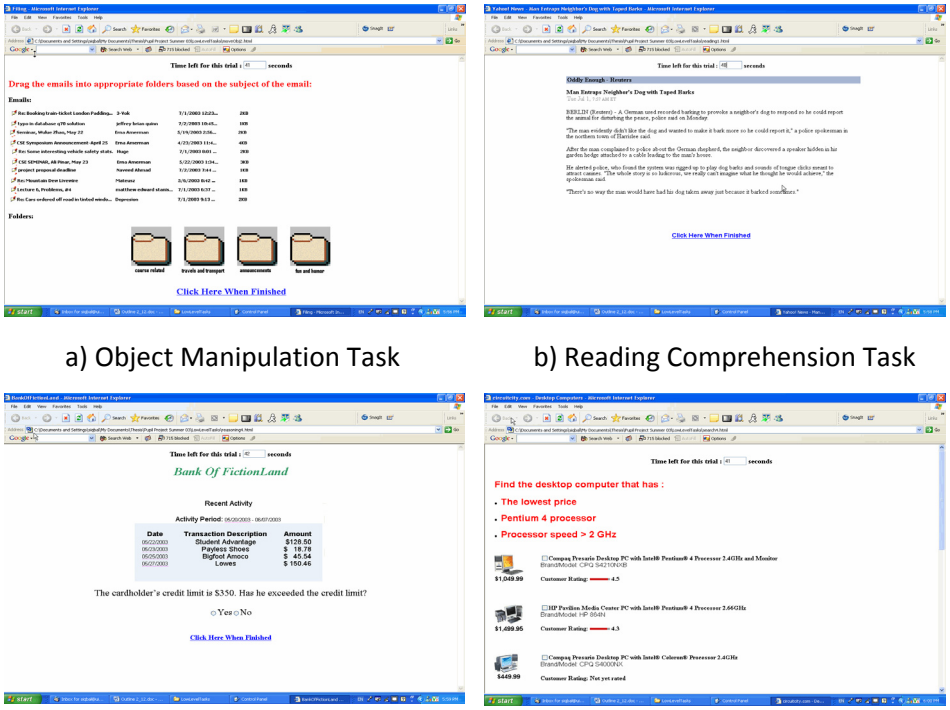
- How well does pupil dilation correlate with the workload induced by interactive tasks with different levels of difficulty?
- Does this correlation hold across several categories of tasks?

4.1.1 Users and Equipment

Twelve users (6 female) participated in the study and the average age was 24 years (SD=3.23). As a user performed the tasks, their pupil data was recorded using a head-mounted eye tracking system (Eyelink II). The eye-tracker sampled the pupil at 250 HZ with spatial accuracy to about 1/100th of a millimeter (for an average 5 mm pupil) using corneal reflection. Lighting and noise levels of the room were well controlled.

4.1.2 Tasks

Based on a literature review, an informal questionnaire to eight users, and our own experience, we developed four representative categories of tasks: Email Classification, Reading Comprehension, Mathematical Reasoning, and Search. Each category had two levels of difficulty – Easy and Difficult. Our expectation was that the difference in task



a) Object Manipulation Task b) Reading Comprehension Task

c) Mathematical Reasoning Task d) Search Task

Figure 4.1: Screenshots of the four task categories used in the first study

workload between the difficulty levels would cause a difference in pupillary response. The four task categories were:

- *Email Classification*. A user had to drag emails and drop them into appropriate folders, based on classification rules; see Figure 4.1(a). For the easier task, the rules were specific such as using the size of the email, e.g., 1K, 2K, or 3K. For the more difficult task, the rules were less specific, requiring each email to be classified by inferring its topic from the subject header, e.g., travel, course related, fun and humor, announcements, etc.
- *Reading Comprehension*. A user read a given text and answered a few questions about its content; see Figure 4.1(b). We used the Fry Formula (Fry 1968) to ensure the texts differed substantially in reading difficulty. The easier task was rated at a grade 9 level while the more difficult task was rated at a grade 17 level.
- *Mathematical Reasoning*. A user performed mathematical calculations; see Figure 4.1(c). For the easier task, a user mentally added two four digit numbers and then selected the correct answer from a list of three choices. For the more difficult task, a user mentally added 4 five-digit numbers, retained the result in memory, and decided whether the result exceeded a given number.
- *Searching*. A user searched for a specific product from a list of similar products given a set of constraints; see Figure 4.1(d). For the easier task, a user had to find the product from a list of seven products given just one constraint, e.g., the cheapest camera. For the more difficult task, a user had to identify the correct product using multiple constraints, e.g., the cheapest 3MP camera with 3X digital zoom.

4.1.3 Procedure

Upon arrival at the lab, the user filled out a background questionnaire and was given general instructions. The eye tracking system was then configured and calibrated. A user performed eight tasks – one easy and one difficult task from each of the four categories. At the beginning of each category, the user received specific instructions and performed a practice. Baseline pupil size was collected by having the user fixate on a blank task screen for a few seconds. The actual tasks were then performed. After completing each task, the user rated its difficulty on a 1-5 scale (1=very easy, 5=very difficult). The presentation order of task category and tasks within each category were randomized.

Users were instructed to perform the tasks as quickly and as accurately as possible. The system logged task performance and screen interaction was recorded for later analysis.

4.1.4 Measurements

A user's subjective rating and task completion time were collected to validate task workload associated with each level of task difficulty. A user's pupil data and on-screen interaction were recorded separately, but were synchronized by correlating timestamps.

For each user, we computed the *percentage change in pupil size* (PCPS), which is the measured pupil size at each sample minus the baseline, divided by the baseline. This calculation is consistent with (Hess 1972) and normalizes for users having different baselines. Also, the easy and difficult task screens were carefully designed to avoid major differences in overall screen luminance. The average PCPS (APCPS) from the beginning to end of each task was used as the task-evoked pupillary response.

4.1.5 Results

A 4 Category (Classification, Comprehension, Reasoning, and Search) x 2 Level of Difficulty (Easy and Difficult) repeated measures ANOVA was performed on the data.

4.1.5.1 Validation of Task Workload

We used task completion time (Figure 4.2) and subjective ratings of difficulty (Figure 4.3) to validate task workload. An ANOVA showed that Category had a main effect on task completion time ($F(3,33)=30.07$, $p<0.0005$). Post hoc tests showed that users spent more

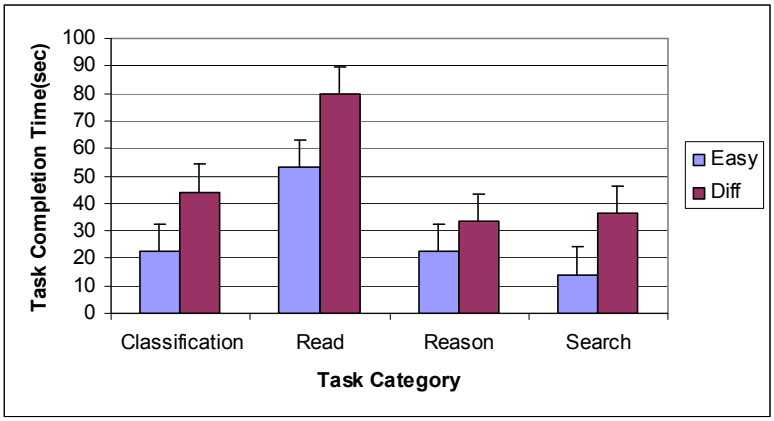


Figure 4.2: Average completion time for each task. Error bars show 95% CI of mean.

time on Comprehension (μ=66.5s) than Classification (μ=33.3s, p<0.001), Reasoning (μ=27.9s, p<0.003) and Search tasks (μ=25.2s, p<0.001). Users also spent more time on

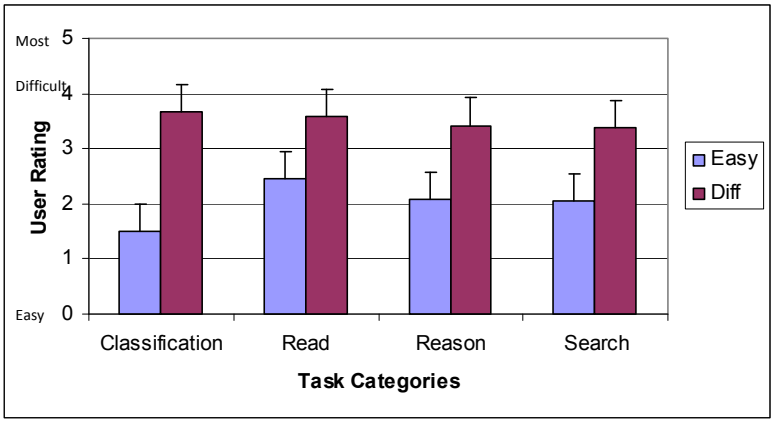


Figure 4.3: Average user rating for each task. Error bars show 95% CI of mean.

Classification than Reasoning (p<0.0005) and Search tasks (p <0.001).

Difficulty also had a main effect on task completion time (F(1,11)=190.9, p<0.0005). Users spent more time on difficult tasks and post-hoc comparisons showed that this effect existed for all but the Reasoning tasks. The interaction between Category and Difficulty (F(3,33)=6.75, p<0.001) was significant, mainly due to the Reading category.

For subjective ratings, Difficulty had a main effect (F(1,11)=34.91, p<0.0005), with higher ratings for the more difficult task in each category. An interaction between Category and Difficulty was detected (F(3,33)=4.12, p<0.014), mainly due to the easier task in the Classification category. There was no main effect of Category.

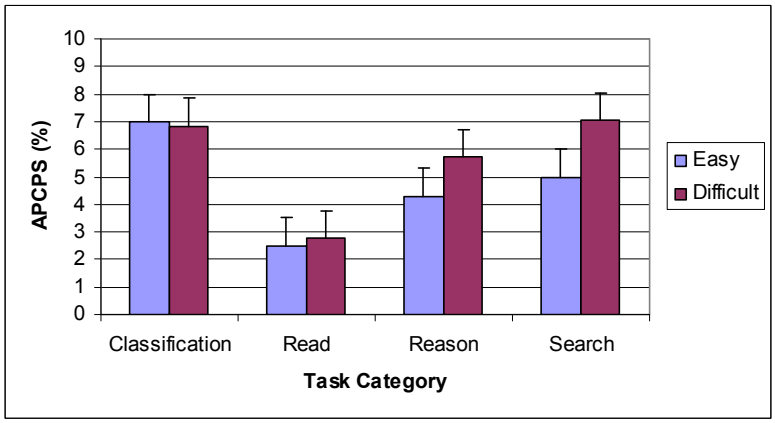
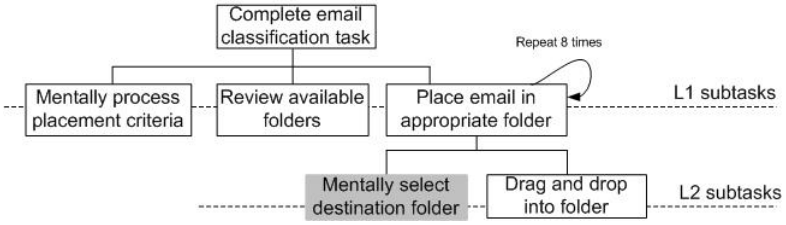
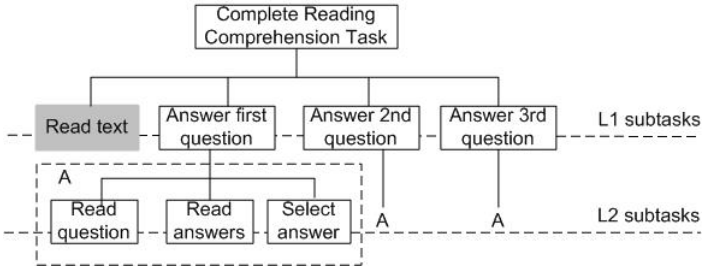


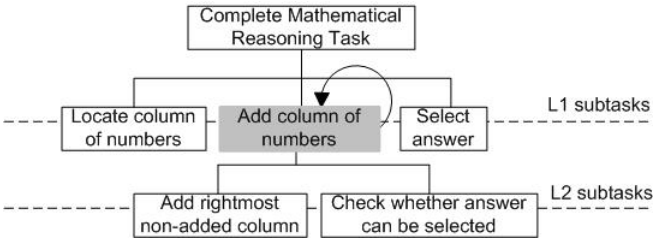
Figure 4.4: Average PCPS for each task. Error bars show 95% CI of mean.



a) Email Classification



b) Reading Comprehension



c) Mathematical Reasoning

Figure 4.5: GOMS Analysis for Email Classification, Reading Comprehension and Mathematical Reasoning

These results validate that the more difficult task had higher task workload than the easier task in each category, and thus a similar pattern is expected for pupillary response.

4.1.5.2 Effects of Task Workload on Pupillary Response

For this analysis, we computed the average percent change in pupillary response (APCPS) for the duration of a task, summarized in Figure 4.4. An ANOVA showed that Category had a main effect on APCPS ($F(3,33)=4.74$, $p<0.007$). This result was not unexpected, as we did not design each category to have the same average task workload.

Surprisingly, Difficulty did not have a main effect ($F(1,11)=3.12$, $p<0.11$). T-tests between the Easy and Difficult tasks within each category revealed a difference only for Search ($p<0.025$). This was inconsistent with our expectation, especially since the ratings and completion times suggested that a difference should exist between difficulty levels.

Except for the Search task, which induced sustained mental effort, the other tasks could be hierarchically decomposed into multiple subtasks. Since only certain subtasks differed in terms of the cognitive effort required, averaging PCPS over the duration of an entire task may dilute the effects of those subtasks. If pupillary response for only those subtasks with different cognitive demands were considered, then the expected differences between levels of difficulty might be found. This was the focus of our second analysis.

4.1.5.3 Effects of Cognitive Subtasks on Pupillary Response

We performed a GOMS analysis to decompose the Classification, Comprehension, and Reasoning tasks into their component goals and operators (Card, Moran et al. 1983), collectively referred to as *subtasks*. The decomposition continued until there was no observable or meaningful separation between subtasks. The models are shown in Figure 4.5. Accuracy was measured by how well the models predicted the interaction sequences in the videos. On average, the models were about 95% accurate and there was no pattern to the errors.

Classification was decomposed into three first-level (L1) subtasks, with the third subtask repeated being eight times. The third L1 subtask was further decomposed into two second-level (L2) subtasks. Between easy and difficult Classification tasks, Select

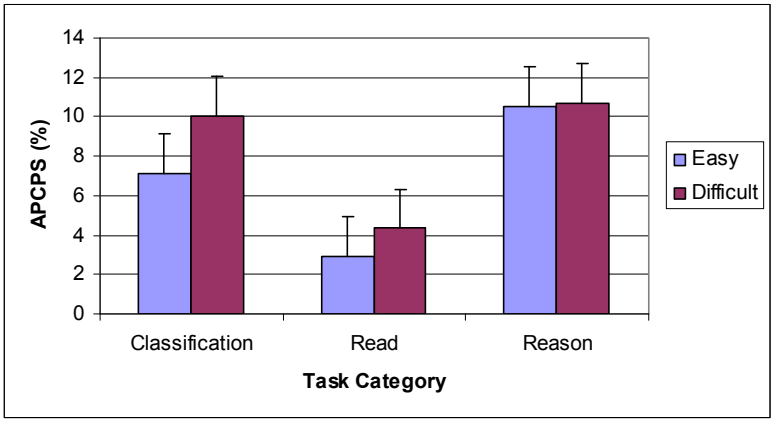


Figure 4.6: Average PCPS for cognitive subtasks. Error bars show 95% CI of means

Folder was the only subtask expected to induce meaningfully different workload. The Reasoning and Comprehension tasks were decomposed using a similar approach. Between easy and difficult Reasoning tasks, Add Column was the only one expected to induce different workload and, for Comprehension, Read Text was the only one expected to induce different workload. APCPS was then calculated using just these subtasks.

Paired t-tests showed that the cognitive subtasks did induce higher pupil dilation than the other parts of the task ($p<0.01$, $p<0.18$, $p<0.001$ for Classification, Comprehension, and Reasoning, respectively). Although the difference for Comprehension was not significant, the trends were in the right direction. These results indicate that pupil dilation is sensitive to the changing workload demands of subtasks during task execution.

Including only the specified subtasks, we performed ANOVAs similar to our first analysis. Difficulty now had an effect on APCPS ($F(1,11)=4.97$, $p<0.048$). As shown in Figure 4.6, the more difficult tasks induced higher APCPS ($\mu=8.36$) than the easier tasks ($\mu=6.85$). Paired t-tests showed that a difference existed between the easy ($\mu=7.14$) and difficult ($\mu=10.03$, $p<0.021$) subtasks for Classification. For Comprehension and Reasoning, differences between easy and difficult tasks were not significant ($p<0.14$ and $p<0.79$), but the trends were in the expected direction. While the easy and difficult tasks for these categories differed along completion time and subjective ratings, the differences were apparently not enough to cause detectable changes in pupillary response.

4.1.6 Discussion

Our results show that pupillary response can be used to reliably measure mental workload for interactive tasks, assuming appropriate environmental controls. For sustained effort tasks, average pupil dilation correlates well with the overall task difficulty. For tasks that have varying task load during execution, pupil dilation demonstrates transient changes that are concomitant with the varying demands of the task, and increased pupil dilation occurs for the more cognitively demanding subtasks. Although measures such as task completion time and user ratings provide overall measures of workload, they do not reflect the changes in workload that a user experiences during task execution (Yeh and Wickens 1988) and, as our results show, these changes can be meaningfully different.

Our results suggest that to better understand how workload changes during tasks with complex structures, pupillary response data should be aligned to the corresponding models of task execution. When comparing workload between tasks, for example, this would allow subtasks that are non-cognitive or that require similar mental effort to be filtered, allowing for a more effective comparison. Also, this would enable investigation of whether there are detectable decreases in workload at subtask breakpoints (Miyata and Norman 1986), where a user has just completed one subtask and begins activating action schema for the next (Altmann and Trafton 2002). This was the focus of our next study.

4.2 Investigating Patterns of Workload Changes during Interactive Tasks

The purpose of the study was to develop further understanding of the relationship between mental workload and the structural characteristics of goal-directed tasks. Our focus was on examining how workload changes at subtask breakpoints within the hierarchical structure of a task's execution, how much this change differs at different levels within the task hierarchy, and how much workload changes among different types of subtasks. Answers to these questions will advance understanding of how to develop methods to identify low cost moments for interruption during interactive tasks.

4.2.1 Experimental Tasks

For the experiment, three interactive tasks were developed:

- *Route planning*. An interactive map was provided that showed two separate routes between two cities marked with start and end symbols (see Figure 4.7). For each route, there were three segments from the source to the destination. A distance and fare were associated with each segment, and were available through a tooltip that appeared when the user moved the cursor over a segment. To perform the task, the user moved the cursor over the first segment in the map corresponding to the first route, committed the distance and fare information shown in the tooltip to memory (the tooltip disappeared when the cursor was moved away), and entered the data into the corresponding row in the table. A user completed each row in the table for the first route, mentally added the distance and fare columns, and entered the results into

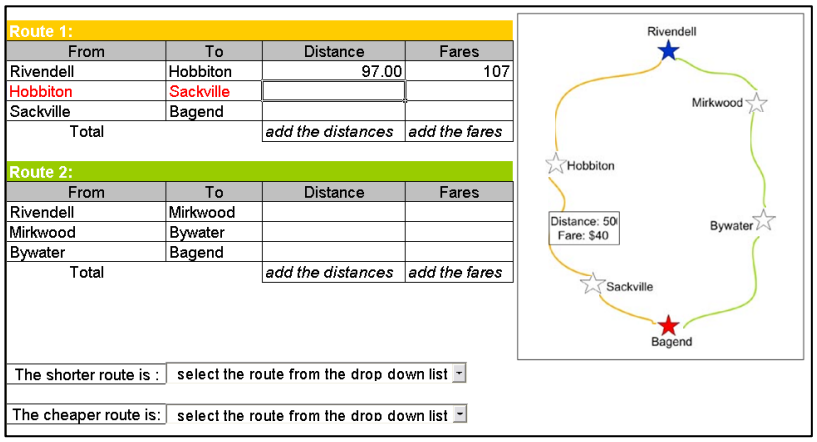


Figure 4.7: The interactive route planning task. A user retrieves distance and fare information from the map, enters the data into the tables, adds the distances and fares, and selects the shorter and the cheaper of the two routes.

the last row. The user then repeated this process for the second table and route. Distance and fare values were manipulated (number of digits) to affect the difficulty of storing and recalling their values from memory as well as computing their sum. After completing both tables, the user selected the shorter and the cheaper of the two routes from drop down lists.

- *Document editing*. A user was given a document with three annotations (see Figure 4.8). The content of the document was about the social hierarchy of a common pet (cats), selected because we felt it would be familiar and understandable to most users.

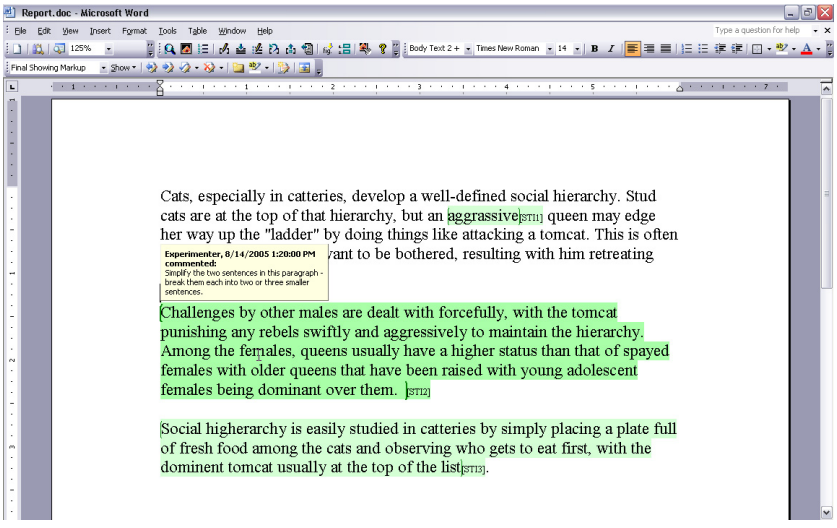


Figure 4.8: The document editing task. A user edited the document based on each of three annotations. Once edited, the document was saved to a specified directory and file name.

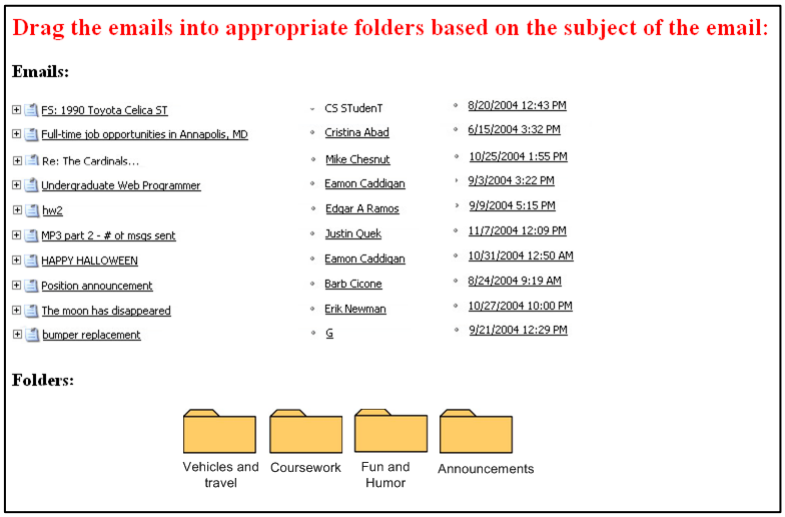


Figure 4.9: The email classification task. Users reasoned about the classification of each email (starting from the top) using its subject descriptor, and then dragged the e-mail into the corresponding folder below. These actions were repeated for each of the emails.

A user edited the document according to each annotation, which appeared as a tooltip when the cursor was moved over the corresponding highlight. After reading an annotation, the user located the corresponding text, made the desired edit, and repeated two more times. The document was saved to a specified directory and file, given a priori. The edits were manipulated to have varying difficulty, e.g., the easiest edit was to correct one misspelled word, the medium edit was to locate and correct two misspelled words, and the most difficult was to rephrase a sentence so that it was grammatically correct.

- *E-mail classification*. For this task (see Figure 4.9), a user was asked to classify a set of nine email messages into a set of supplied categories; e.g., coursework, vehicles and travel, announcements, and fun and humor. The user would review the subject descriptor of a message, reason about which category it belonged to, and drag the message into the corresponding folder. The user then repeated this sequence for the remaining messages. The content of the subject descriptors was manipulated to affect the difficulty of the classification subtask, e.g., some descriptors had the name of the destination category within it while others were more ambiguous.

These tasks were carefully designed to have meaningful subtasks of varying difficulty, well-defined breakpoints between subtasks, a representative sample of interaction, and a

prescribed execution sequence. A prescribed sequence was necessary to be able to align each user's workload data to the corresponding model of task execution. The lower-level cognitive subtasks, e.g., memory store and recall, comprehension, and reasoning are representative of those within many other tasks. Though the tasks are relatively simple, it is important to note that they are more complex and of longer duration than tasks used in many prior experiments involving pupillary response, e.g., see tasks used in (Bradshaw 1967; Kahneman 1967; Juris and Velden 1977; Hytintk, Tommola et al. 1995; Takahashi, Nakayama et al. 2000).

4.2.2 Users and Equipment

A total of 24 users (7 female) participated in the experiment, with ages ranging from 19 to 50 (M=25.4). All users had normal or corrected-to-normal vision. As users performed tasks, their pupil data was recorded using a head-mounted eye tracking system (Eyelink II). The eye-tracker sampled the pupil at a high temporal frequency of 250 HZ with spatial accuracy to about $1/100^{th}$ of a millimeter using corneal reflection. Lighting and noise levels of the task environment were well controlled. Twelve users performed both the route planning and document editing tasks while the remaining twelve performed the email classification task. This reduced the time that any one user had to wear the eye tracking equipment, but did not impact the results as each task was analyzed separately.

4.2.3 Procedure

Upon arrival at the lab, we went through an informed consent process with the user and provided general instructions for the tasks. After questions were answered, we set up the eye-tracker and calibrated the system. At the start of the session, the user was given specific instructions and performed practice tasks. Just before each experimental task, we collected baseline pupil size by having the user fixate on a blank task screen for a few seconds. The user was asked to perform the tasks as quickly and accurately as possible. Time-stamped samples of pupil data were logged to a file while the user's screen interaction was recorded with eye gaze overlaid. Because the videos and pupil data received time stamps from the same clock, we could precisely align the two data sets. The entire experimental session lasted about 30 minutes.

4.2.4 Task Models and Validation

Figure 4.10, Figure 4.11 and Figure 4.12 show the task models for the Route Planning, Document Editing, and Email Classification tasks, respectively, reusing repetitive parts for brevity. The term *subtask* refers to any node in the model and *subtask breakpoint* refers to the period between adjacent subtasks. *Level of breakpoint* between two adjacent subtasks is 1 + the depth of their shared ancestor in the model. For example, in Figure 4.9, consider the "Locate segment" and "Store data" subtasks at the left of level 4. When a user completes the "Locate segment" subtask and moves to "Store data", this defines a level 4 breakpoint, since the depth of their shared ancestor "Retrieve segment" is (1 +) 3. When a user completes the "Store data" subtask and moves to "Recall", this defines a level 3 breakpoint, since the depth of their shared ancestor "Enter data for segment 1" is (1 +) 2. Finally, *subtask type* refers to whether the subtask represents a memory store, memory recall, reasoning, language comprehension, language generation, or motor operator.

The task models were developed in an iterative manner. For each task, we developed an initial model through our own analysis of the task's execution. The initial models were refined based on screen interaction videos of four users performing the tasks prior to and independent from the reported experiment. The interaction sequences predicted by the leaves in our task models were compared to the sequences observed in the interaction videos, and our models were refined until a high degree of agreement was reached.

We measured the accuracy of our final task models by comparing the operators in the models to the observable events (keyboard, mouse, and eye gaze) in the interaction videos recorded during the experiment. An error step was defined to be a deviation from the prescribed sequence. If the user committed an error, each action after that step would count as an error until the user again performed a step in the prescribed sequence, from which point the analysis continued as discussed in (Card, Moran et al. 1983).

The final task model for Route Planning had 4 levels and 81 nodes. The average error rate was 2.81% with no detectable pattern to the errors. Repeating this same process for Document Editing, the resulting model had 4 levels and 38 nodes with an average error rate of 2.3%. The model for Email Classification had 2 levels and 25 nodes and matched users' execution of the task without error.

4.2.5 Measurements

Following prior work (Hess 1972), workload was calculated as the percent change in pupil size (PCPS) for each sample of data relative to the baseline. Eye blinks, which were identified by the eye tracking system, were accounted for by linearly interpolating the missing values (Verney, Granholm et al. 2001). For each subtask and breakpoint, we also computed the average PCPS (APCPS) for that region of data. The duration of subtasks ranged from about 25ms for the lowest-level subtasks, to about 1 min for the higher level subtasks, to about 5 min for the root node (the entire task). The duration of breakpoints ranged from about 8ms to 6 seconds ($\underline{M}$=487ms, S.D.=574ms), with higher level breakpoints generally being of longer duration. A more detailed analysis of the durations of the breakpoints within each task will be provided within the Results section.

4.2.6 Alignment and Analysis

As the models accurately reflected a user's execution sequences in the tasks, we were able to precisely align a user's pupillary response to the task models. Since each user performed the tasks at different speeds, our approach was to align the pupil data to the subtasks in the model, not to time, starting from the leaf operators and working upwards.

For each leaf subtask, we identified the beginning and end time stamp from the screen interaction video and used these timestamps to index the pupillary response file. The corresponding PCPS data was then extracted and associated with that subtask. APCPS for higher-level subtasks was calculated by averaging the PCPS values associated with their child subtasks, and this process was repeated until the root node was reached. For breakpoints, we extracted the PCPS data from the end timestamp of the preceding subtask to the begin timestamp of the subsequent subtask and computed APCPS values as before. To account for latency in the response of the pupil to the onset of a stimulus, we temporally shifted the pupil data by a small amount (500 msecs) prior to aligning it with the task model (Kramer 1991). To compute decrease in workload at a breakpoint, we compare the APCPS of the breakpoint to the APCPS of its preceding subtask. This was to ensure that we are offering a fair comparison between these two regions of data. Finally, when a non-leaf subtask precedes a breakpoint, we compare the APCPS of the breakpoint to the APCPS of the *last leaf operator* of that subtask. For example, in Figure 4.10, the

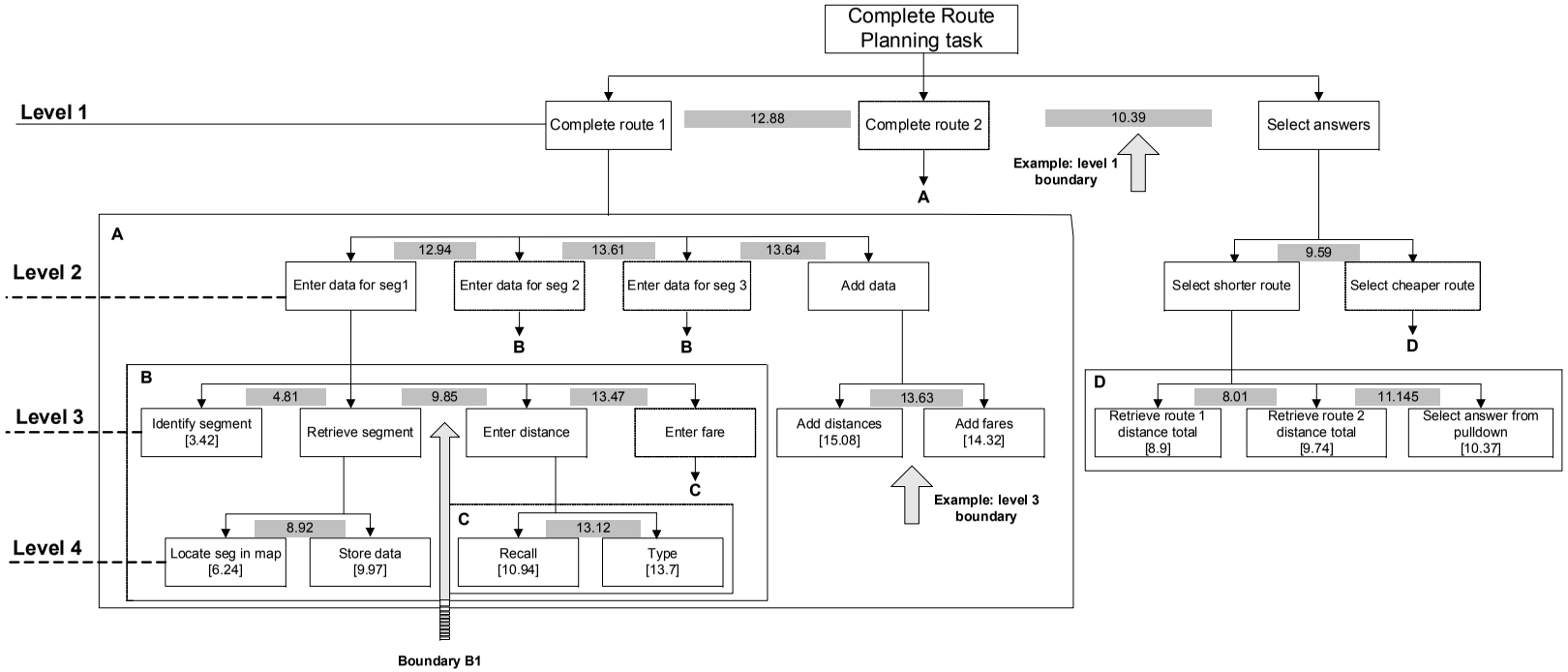


Figure 4.10: A workload aligned task model for Route Planning. The interior nodes represent goal nodes, the leaf nodes represent operators, and time moves from left to right. Regions A, B and C show parts of the task repeated elsewhere in the model. Within each sub subtask, we provide the [APCPS] for that subtask. Each shaded area indicates a breakpoint and contains the [*APCPS*] across it.

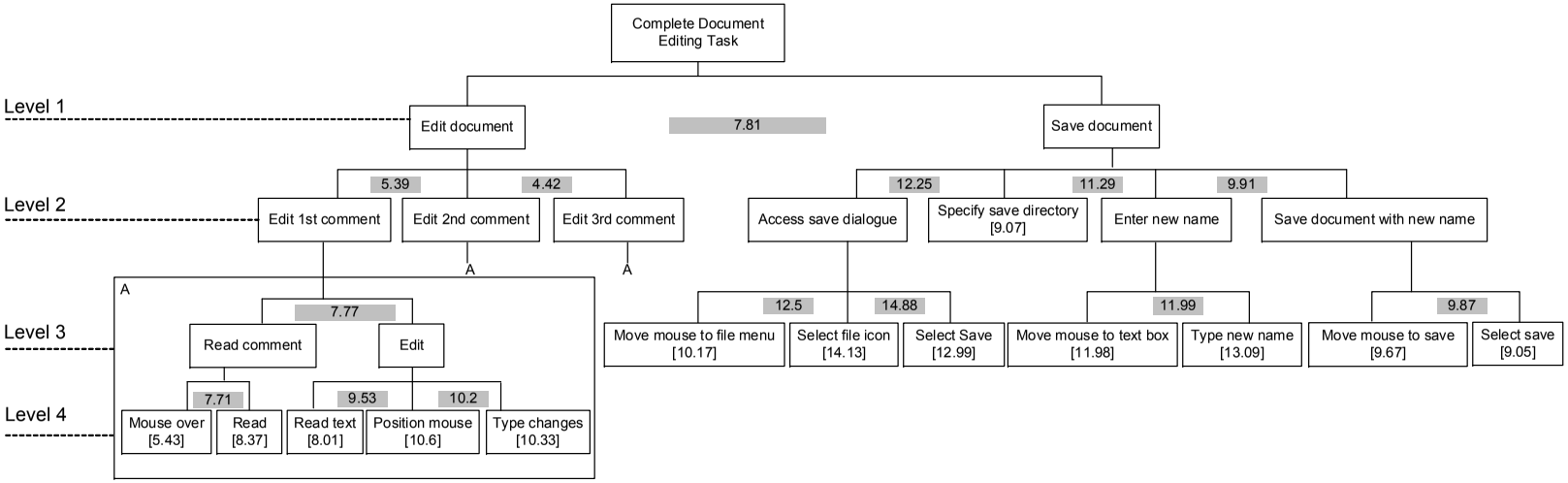


Figure 4.11: A workload aligned model for Document Editing. The interior nodes represent goal nodes, the leaf nodes represent operators, and time moves from left to right. Regions A shows parts of the task repeated elsewhere in the model.

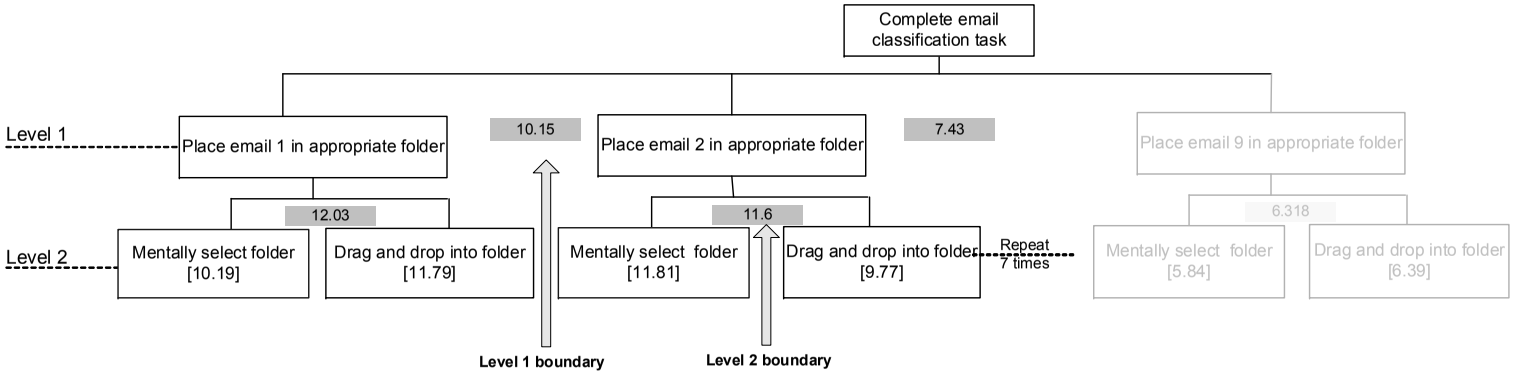


Figure 4.12: A workload aligned task model for Email Classification. The interior nodes represent goal nodes, the leaf nodes represent operators, and time moves from left to right. The level 1 subtask is repeated 9 times. Within each subtask, we provide the [APCPS]

decrease at breakpoint B1 is computed as the difference in APCPS between B1 and the operator *Store Data* (the last operator of *Retrieve segment*). This was done to ensure that earlier parts of longer subtasks were not unfairly affecting the comparison.

4.3 Results

To provide further confidence in our measure of workload, we first check that regions within the execution structure of the tasks that were expected to induce lower/higher workload did indeed correspond to lower/higher values of PCPS. Then, for each task, we report results of how the different types of subtasks affected workload, how the level within a task model affected workload at the subtask breakpoints, and how much workload differed between a breakpoint and its preceding subtask.

The reader should keep in mind that small changes in pupillary response can represent meaningful changes in workload, but that there is also an upper bound on how much a user's pupil size will increase due solely to increases in mental processing effort.

4.3.1 Validation of Workload Measure

To validate our workload measure, we compared pupillary response between different regions of the tasks that would presumably require different amounts of mental processing effort. For Route Planning, we performed an ANOVA with Load (fewest, middle, and most digits) as the factor on the APCPS of Recall subtasks. Results showed that Load had a main effect on APCPS ($F(2,46)=6.24$, $p<0.01$), where Recall subtasks that required more digits to be retrieved from memory had higher APCPS. For Email Classification, an ANOVA with Classification Difficulty (easier, more difficult) as the factor showed that the more difficult classification (where more mental effort was required to select the target folder for the e-mail message) had higher APCPS than for the easier classification ($F(1,7)=9.81$, $p<0.05$). For Document Editing, Difficulty (simple, medium, and difficult edit) did not have a main effect, though the trends were in the expected direction. We attribute this lack of significance to the three types of edits being closer in terms of the mental effort required relative to the subtasks being compared for the other two tasks. Overall, these results confirm that users' pupil size was changing in response to the changing difficulty of the subtasks.

4.3.2 Route Planning

Figure 4.13 shows the average (across users) APCPS for each leaf (operator) subtask and all breakpoints within the model for Route Planning. In the graph, note how workload rises quickly at the onset of the task and then rises and falls throughout execution. Inspection of the graph clearly shows transitory decreases in workload at the two level 1 breakpoints.

4.3.2.1 Workload During Subtasks

To test whether performing subtasks induced workload over the baseline value, we performed a t-test on the APCPS values of the subtasks. Our analysis included only those subtasks that required cognitive effort such as storing, recalling, or reasoning about distance and fare information, rather than motor subtasks, as the relationship between cognitive effort and pupillary response is the one best established (Beatty 1982). Results showed that APCPS was greater than 0 across subtasks ($\underline{M}$=11.98, $\underline{SD}$=7.6, $t(263)=25.75$, $p<0.001$). This represents about a 12% increase over the baseline value and shows that the subtasks did impose increased workload on a user.

An ANOVA with Subtask (Store, Recall, and Reasoning) as the factor showed a main effect on APCPS ($F(2,261)=4.87$, $p<0.01$). Post hoc tests showed that Reasoning induced more workload than Store (difference was 4.3 percentage points, $p<0.01$) and Recall (difference was 3.2 percentage points, $p<0.05$), while there was no difference found between Store and Recall subtasks.

4.3.2.2 Workload at Subtask Breakpoints

A t-test showed that APCPS at breakpoints was greater than 0($\underline{M}$=11.68, $\underline{SD}$=7.62, $t(611)=37.91$, $p<0.001$) and that Level had a main effect on the APCPS of breakpoints ($F(3,608)=2.61$, $p<0.05$). Post hoc tests showed that the APCPS of breakpoints at Level 3 (M=10.78) was less than the APCPS of breakpoints at Level 4 ($\underline{M}$=12.59, $p<0.05$). Other pairs were not significant, though the means were in the expected direction ($\underline{M}$=11.64 for Level 1 and $\underline{M}$=12.37 for Level 2). Among all breakpoints in the task model, the Level 2 breakpoint between *Retrieve route 1 total* and *Retrieve route 2 total* had the lowest APCPS (9.57) while the Level 3 breakpoint between *Add distances* and *Add fares* had the

highest (13.99). This indicates that the workload carried through a breakpoint depends not just on the level in a model, but also on the mental demands of the surrounding subtasks.

The overall average duration of the breakpoints was 590ms. Level had a main effect on breakpoint duration ($F(3,593)=11.26$, $p<0.001$). Post hoc tests showed that level 1 breakpoints ($\underline{M}$=887ms) were of longer duration than level 3 ($\underline{M}$=456ms, $p<0.001$) and level 4 ($\underline{M}$=483ms, $p<0.01$) breakpoints, and that level 2 breakpoints ($\underline{M}$=691ms) were of longer duration than level 3 ($p<0.001$) and level 4 ($p<0.001$) breakpoints.

4.3.2.3 Decrease of Workload at Subtask Breakpoints

Breakpoint Decrease is computed as the difference between APCPS at the breakpoint and the preceding subtask. With values for all breakpoints included, a t-test showed that Breakpoint Decrease was slightly greater than 0, but did not reach a level of significance ($\underline{M}$=0.029, $\underline{SD}$=3.91, $t(612)=0.18$, $p=0.85$). This indicates that not all breakpoints exhibit

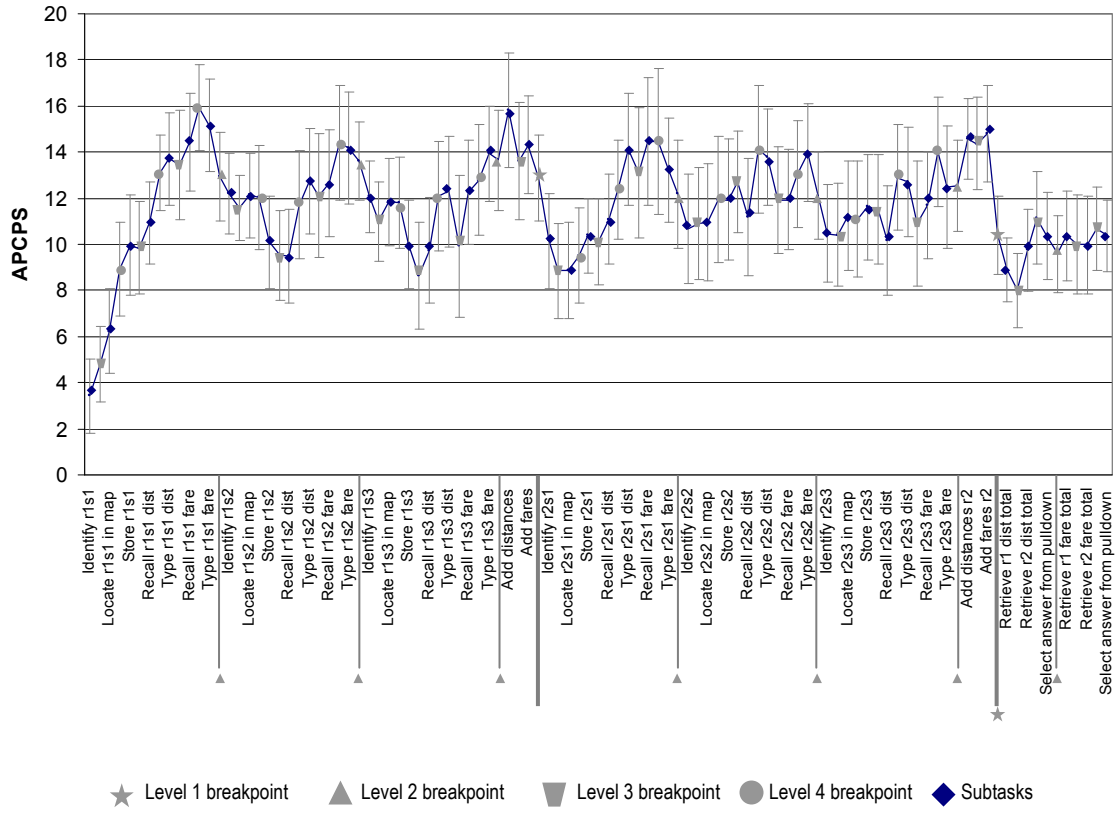


Figure 4.13: APCPS of the leaf subtasks and all breakpoints within the model for Route Planning. Vertical lines within the subtask labels demarcate Level 1 and 2 breakpoints.

a detectable decrease in workload, likely because the numerous lower level breakpoints were closely related and had a high degree of mental carryover.

However, when the lowest level breakpoints (Level 4) are excluded, the same analysis now shows Breakpoint Decrease to be greater than 0 ($\underline{M}$=0.7905, $\underline{SD}$=3.8, $t(397)=4.15$, $p<0.001$). This effect continues to become stronger as lower-level breakpoints are successively excluded from the analysis. This result indicates that workload temporarily decreases as a user crosses through a breakpoint during execution of a task, but the result only holds for breakpoints that are above a certain level (depth) within the task model.

Exploring this pattern further, we found that Level had a main effect on Breakpoint Decrease ($F(3,608)=18.42$, $p<0.001$). Post hoc tests showed that decreases at level 1 were greater than at level 2 (1.95 percentage points), level 3 (2.38 percentage points, $p<0.05$) and level 4 (4.3 percentage points, $p<0.001$). Level 2 decreases were greater than level 3 (0.43 percentage points) and level 4 (2.34 percentage points, $p<0.001$), and Level 3 decreases were greater than level 4 (1.91 percentage points, $p<0.001$).

Overall, this pattern shows that workload tends to decrease more at breakpoints higher in the task model than at breakpoints lower in the model. We also found that workload changed *within* the same level in a task model. For example, APCPS between the two level 1 breakpoints was different ($F(1,22)=5.31$, $p<0.05$) with an absolute difference of 2.96 percentage points.

4.3.3 Document Editing

Figure 4.14 shows the average APCPS for each leaf subtask and all breakpoints within the model for the Document Editing task. As in the Route Planning task, workload rises at the onset of the task, rises and falls throughout task execution, and temporarily decreases at salient breakpoints within the task, i.e., after completing each of the three edits.

4.3.3.1 Workload During Subtasks

Including only cognitive subtasks (language comprehension and generation, and recall), a t-test showed that APCPS for subtasks was greater than 0 (M=8.62, SD=7.35,

t(111)=12.41, p<0.001). This shows an 8.62% increase over the baseline, meaning that the subtasks did induce increased workload, but not as much as in the route planning task.

An ANOVA with Subtask (Comprehension, Generation, and Recall) as the factor showed a main effect on APCPS (F(2,109)=4.19, p<0.05). Recall induced more workload than Comprehension (difference was 5.9 percentage points, p<0.05) and Generation (difference was 3.04), and Generation induced higher workload than Comprehension (difference was 2.86). These results are consistent with Route Planning where different types of subtasks also induced different amounts of workload.

4.3.3.2 Workload at Subtask Breakpoints

A t-test showed that APCPS of breakpoints was greater than 0 (M=9.02, SD=8.4, t(233)=16.43, p<0.001). Level did not have a main effect, but the trends were in the expected direction (M=7.82 for Level 1, M=8.22 for Level 2, M=9.40 for Level 3, and M=9.26 for Level 4). Among all breakpoints, the Level 2 breakpoint between *Edit second comment* and *Edit third comment* had the lowest APCPS (4.42), while the highest APCPS

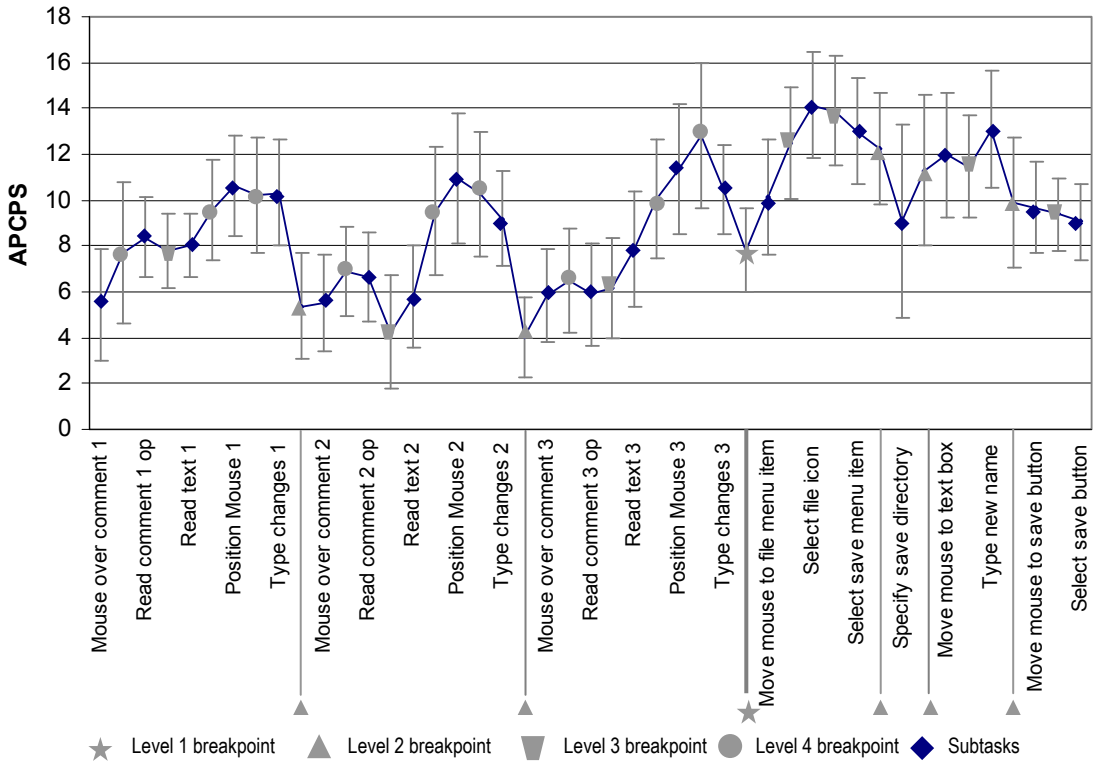


Figure 4.14: APCPS of leaf subtasks and all breakpoints within the model for Document Editing. Vertical lines within the subtask labels differentiate Level 1 and 2 breakpoints.

was at the Level 3 breakpoint between *Select file menu* and *Select save* (14.99).

The overall average duration of breakpoints was 528ms. Level had a main effect on the duration of a breakpoint ($F(3,241)=5.31$, $p<0.001$). Post hoc tests showed that breakpoints at level 1 ($\underline{M}=1.1$s) were of longer duration than breakpoints at levels 3 ($\underline{M}=461$ms, $p<0.05$) and 4 ($\underline{M}=401$ms, $p<0.01$), and breakpoints at level 2 ($\underline{M}=746$ms) were of longer duration than those at level 4 ($p<0.05$).

4.3.3.3 Decrease of Workload at Subtask Breakpoints

With all breakpoints included, a t-test did not show Breakpoint Decrease to be greater than 0. As with the Route Planning task, excluding the lowest level breakpoints (Level 4) and re-running the t-test now showed Breakpoint Decrease to be greater than 0 ($\underline{M}=1.34$,

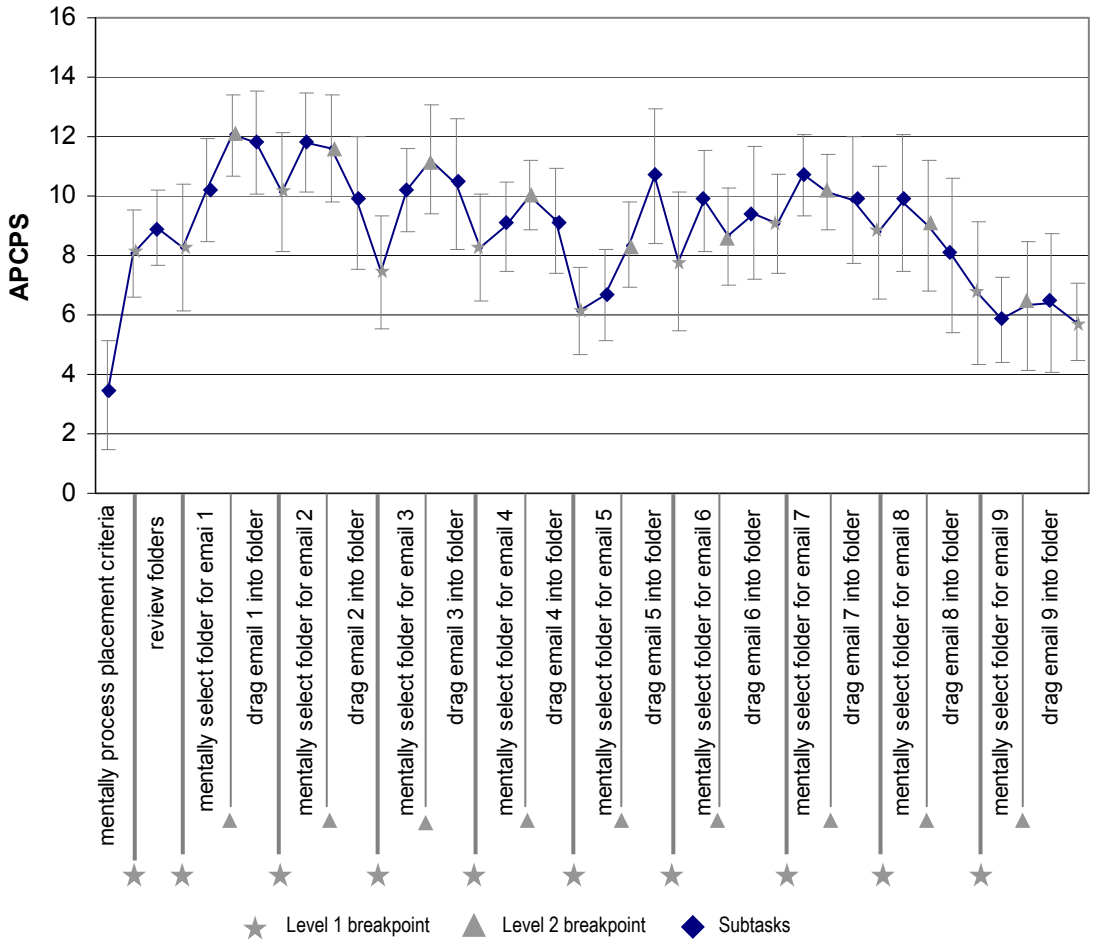


Figure 4.15: APCPS of leaf subtasks and all breakpoints within the model for Email Classification. Vertical lines within the labels demarcate Level 1 and 2 breakpoints.

SD=4.01, t(133)=3.87, p<0.001). An ANOVA with Level as the factor showed a main effect on the APCPS of breakpoints (F(3, 234)=16.81, p<0.001). Breakpoint Decrease at Level 1 was similar to Level 2, but larger than level 4 (3.99 percentage points, p<0.01). Level 1 also had quantitatively higher decrease than level 3 (2.61 percentage points), but the difference did not reach a level of significance. Breakpoints at level 2 were found to have a larger decrease than at level 3 (2.94 percentage points, p<0.001) and level 4 (4.32 percentage points, p<0.001). Breakpoints at level 3 tended to have a larger decrease than at level 4 (1.39 percentage points), but did not reach significance. Overall, this pattern of results shows that workload tends to decrease more when a breakpoint higher in the model is reached during an interaction sequence, consistent with results from Route Planning.

4.3.4 Email Classification

Figure 4.15 shows the average APCPS for subtasks and all breakpoints within the model for Email Classification. Analogous with the other tasks, the graph shows a temporary decrease in workload at the top-level breakpoint corresponding to the completion of the classification of each mail message. Also, note that the structure of this task is simpler than the previous tasks, as there were only two levels in the task model and only one type of cognitive subtask – reasoning about the destination folder.

4.3.4.1 Workload During Subtasks and at Breakpoints

For the cognitive subtasks (reasoning), a t-test showed that APCPS was greater than 0 (M=9.48, SD=5.6, t(71)=14.45, p<0.001). This represents a 9.48% increase over the baseline and shows that the subtasks did induce increased workload, as with the previous two tasks. In addition, a t-test showed that the APCPS of breakpoints was greater than 0 (M=8.93, SD=6.05, t(142)=17.65, p<0.001), with Level 1 breakpoints having lower APCPS (M=7.93) than the Level 2 breakpoints (M=9.92). Breakpoints at level 1 (M=484ms) were of longer duration than those at level 2 (M=320ms; F(1, 167)=7.41, p<0.01), and the overall average duration for a breakpoint was 405 ms.

4.3.4.2 Decrease of Workload at Subtask Breakpoints

A t-test showed that Breakpoint Decrease was greater than 0 (M=0.6816, SD=3.73, t(142)=2.19, p<0.05). An ANOVA showed that Level had a main effect on Breakpoint

Decrease ($F(1,141)=14.41$, $p<0.001$), with the decrease at Level 1 breakpoints ($\underline{M}=1.82$, S.D.=2.94) being larger than at Level 2 breakpoints ($\underline{M}= -0.44$, S.D.=4.08). These results show that workload decreased at breakpoints and that it decreased more at breakpoints higher in the task model, which is consistent with results from the other two tasks.

4.3.5 Discussion

The purpose of this study was to better understand how workload changes during execution of goal-directed tasks. Here we summarize our primary findings, situate these findings within resource theories of human attention, and discuss implications of these findings for systems that reason about when to interrupt users engaged in tasks.

First, our results provide further evidence showing that a user's mental workload changes throughout execution of goal-directed tasks. From the perspective of resource theories of attention (Kahneman 1973; Wickens 1980; Wickens 1991; Wickens 2002), this result indicates that the executive system does not statically allocate attentional resources at the onset of a task stimulus, but dynamically allocates and releases resources throughout its execution.

Second, we found that transitory decreases in workload are experienced as subtask breakpoints are reached during task execution. This result is summarized in Figure 4.16.

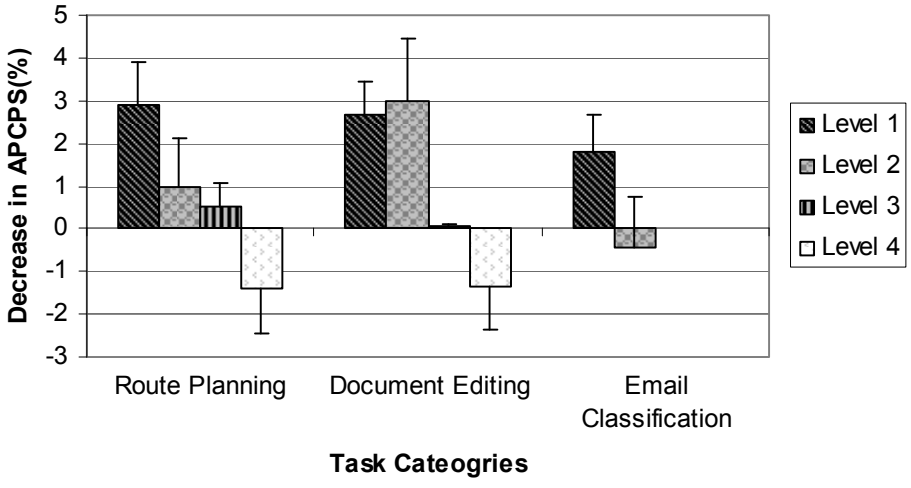


Figure 4.16: Decrease in the APCPS of breakpoints, shown as a function of level and task. Higher level breakpoints (1 and 2) show larger decreases in workload than lower level breakpoints (3 and 4), which exhibited little or no decrease.

A plausible explanation is that the executive system releases attentional resources allocated for the just completed subtask, but has not yet acquired resources for the subsequent subtask. An important implication of this result is that it establishes the principle of using *defer-to-breakpoint* policies for reducing costs of interruption caused by notifications. For example, if notifications could be deferred until a breakpoint is reached, the executive system would have more resources available for performing the interrupting task (Rubinstein, Meyer et al. 2001). Such policies would also be beneficial because breakpoints typically represent moments between explicit interactions with a system. For example, this would prevent notifications from being delivered during text entry or other motor subtasks, even if workload is found to be lower. This finding is consistent with design recommendations from programmers who were interrupted during complex coding tasks in (Fogarty, Ko et al. 2005).

Third, our results showed that the transitory decrease in workload tended to be larger when breakpoints higher in a model were reached during task execution. In addition, higher-level breakpoints were found to be of longer duration than lower-level breakpoints. These results indicate that more resources are released when more salient breakpoints are reached during a task. Whereas, when lower-level breakpoints are reached, the amount of resources released is apparently small, possibly due to cognitive chunking of repetitive or skilled actions (Newell and Rosenbloom 1981) or to large carryover of information being actively maintained in working memory. The implication of this finding is that interruption management systems should favor breakpoints that represent more salient breaks during a user's ongoing interaction, as these should result in lower costs of interruption. In addition, systems need not consider breakpoints that are lower in the task model (roughly beyond the third level) as these appear to provide little benefit over non-breakpoint moments.

Finally, the level of a breakpoint in a task model cannot always predict whether there would be a larger decrease or lower absolute value of workload at a breakpoint. For example, in the document editing task, the breakpoint with the lowest workload was between the second and the third edits, which was not a top level breakpoint. Similarly, for route planning, the lowest workload breakpoint was between selecting the shorter and the cheaper of the routes, which also was not a top-level breakpoint. A plausible

explanation is that the executive system may be maintaining information in short-term memory or prospectively allocating resources in anticipation of a subsequent subtask across some breakpoints, but not others (Trafton, Altmann et al. 2003).

However, although workload varied based on the location of the breakpoint in the hierarchical task model, we were yet to understand the effects of interrupting at breakpoints with different workload. This was the focus of the next study.

4.4 Effects of Interrupting at Breakpoints with Different Workload

Our third study compared effects of interrupting task execution at breakpoints with lower workload, breakpoints with higher workload, and random moments (simulating today's interface). This would allow us to empirically demonstrate the relationship between workload and effects of interruption.

From the workload-aligned models developed in our previous studies, we selected breakpoints with the lowest and highest workload. Our expectation was that scheduling interruptions to occur at the Best moments (low workload breakpoints) would have less negative impact than interrupting at the Worst (high workload breakpoints) and random moments. A no-interruption condition was also included as a control.

4.4.1 Experimental Design

A repeated measures design was used with Timing (Best, Worst, Random, None) and Task (Route Planning, Document Editing, Email Classification) as factors. Eye tracking equipment was not used in this experiment.

4.4.2 Users and Tasks

Twelve users (6 female) participated in the study and ages ranged from 21 to 42. The experiment used the Route Planning and Document Editing tasks from our second study and the Email Classification task from our first study. To use the latter task, we performed enough of the workload alignment such that breakpoints with the lower and higher workload could be identified. While more tasks could have been used, we felt that using three primary tasks provided a reasonable sample while keeping the length of the experiment practical.

We developed similar tasks for each category, but were careful to design them such that they would not induce workload patterns largely different from the existing models. For the interrupting task, users read a news article and selected the most appropriate title from three choices. The interrupting task was adapted from (Adamczyk and Bailey 2004).

4.4.3 Selected Moments for Interruption

For Route Planning, Best was between completing the second route and selecting the shorter (or cheaper) route. Worst was between recalling and entering information into any cell of the table. For Document Editing, Best was between the completion of the last edit and accessing the Save menu. Worst was between positioning the mouse at the intended location and entering changes to the document text. For Email Classification, Best was between placing an email into a folder and preparing to access the next email. Worst was between selecting an email and starting to drag it towards the destination.

Best and Worst for each task were selected by ordering the breakpoints according to increasing APCPS and selecting the breakpoint with the highest and lowest workload. An ANOVA showed that mental workload was lower at the Best moments (μ=7.85) than the Worst moments (μ=12.46; $F(1,11)=9.31$, $p<0.01$) across tasks. For all tasks, Best existed at a level higher in the task model than Worst.

4.4.4 Experimental Setup

Delivery of interrupting tasks used a Wizard of Oz model. The experimenter observed a user's task execution using a RealVNC client connected over a high-speed LAN to minimize latency. At pre-defined moments, the experimenter used custom software to send an interrupting task to a user. Best and Worst moments were defined from the task models and Random moments were delivered at times randomly selected from an interval based on average task completion times.

4.4.5 Procedure

Before each category, specific instructions were given to the user and a practice task was performed. For each category, users performed four task trials, one for each timing condition. Users were instructed to attend to an interrupting task as soon as it appeared and, once complete, resume the primary task. The interrupted task was presented in a

modal window and covered the main work area of the primary task. Users were instructed to complete the tasks as quickly and accurately as possible. After each task trial, users completed the NASA TLX and scales for annoyance and respect. The order of the categories, tasks, and timing conditions were randomized. The study lasted an hour.

4.4.6 Measurements

In the study we measured the following:

- *Subjective workload*. This was measured using the NASA TLX (Hart and Staveland 1988). Users responded by marking a vertical line along continuous scales from low to high. Scores were weighted equally and combined into a single workload value.
- *Resumption lag*. This is the time needed to meaningfully resume the primary task after interruption. Lag was measured as the time from closing the interrupting task window to the first keyboard or mouse action in the primary task in direction of the task goal.
- *Annoyance*. This was measured on a continuous scale from low to high, similar to the TLX. Annoyance was used as a measure of the user's affective state.
- *Respect*. Users rated how respectful the interrupting system was to the primary task, i.e., social attribution. This measure was included because studies show that users often respond to interactive systems socially (Nass, Steuer et al. 1994).

Combinations of these measures have been used to measure effects of interruption in many prior studies, e.g., (Bailey, Konstan et al. 2001; Trafton, Altmann et al. 2003; Adamczyk and Bailey 2004).

4.4.7 Results

Two-way ANOVAs (Task x Timing) were used to analyze the dependent measures.

4.4.7.1 Subjective Workload

Task had a main effect on subjective workload ($F(2,22)=22.01$, $p<0.001$). Post hoc analysis showed that Route Planning ($\mu=2.74$) induced higher subjective workload than both Document Editing ($\mu=2.25$, $p<0.002$) and Email Classification ($\mu=1.77$, $p<0.001$), while Document Editing induced higher workload than Email Classification ($p<0.003$). Timing did not influence subjective workload and there were no interactions. This shows

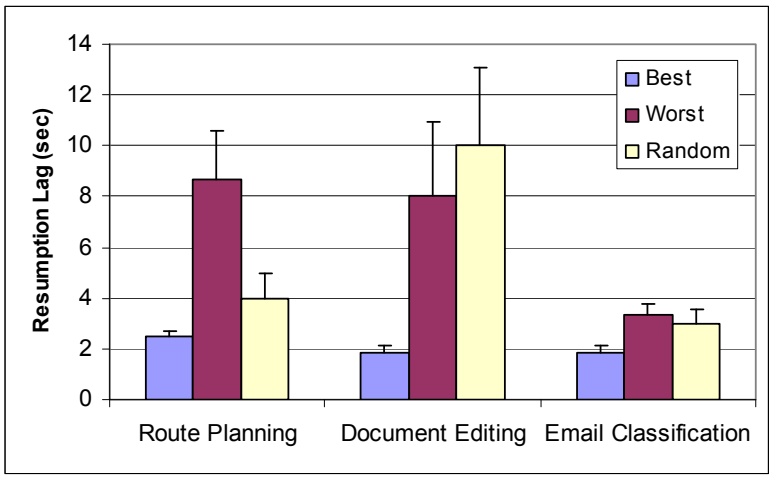


Figure 4.17: Time to resume a primary task after being interrupted in each timing condition

that the interrupting task did not induce subjective workload beyond that of the primary task – regardless of timing. However, the results do allow the tasks to be rank ordered based on workload (Route Planning > Document Editing > Email Classification), which can be used to help interpret later results.

4.4.7.2 Resumption Lag

Figure 4.18 shows results for resumption lag. Task had a main effect ($F(2,22)=6.27$, $p<0.007$). Post hoc analysis showed that users resumed Email Classification tasks faster ($\mu=2.73$s) than Route Planning ($\mu=5.04$s, $p<0.012$) and Document Editing tasks ($\mu=6.63$s, $p<0.004$) after being interrupted. No other differences were found. This is roughly

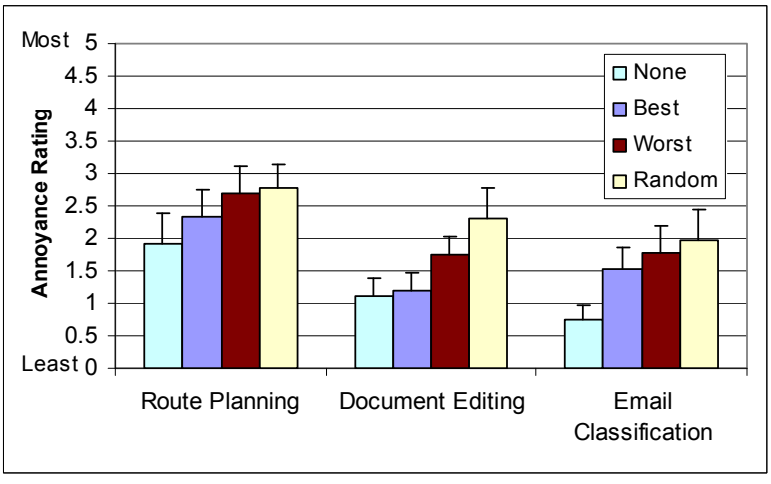


Figure 4.18: Ratings of Annoyance

consistent with the workload ratings, as the lowest workload task had the least resumption lag.

Timing had a main effect ($F(3,33)=6.87$, $p<0.005$). Post hoc tests showed that users resumed the primary task almost 3 times faster after being interrupted at Best ($\mu=2.06$s) than at Worst ($\mu=6.69$s, $p<0.03$) and Random ($\mu=5.66$s, $p<0.007$) moments. This result may be explained by the executive system needing to acquire fewer attentional resources to resume a primary task interrupted at a moment of lower workload (Wickens 2002). No other differences were found and there were no interactions.

4.4.7.3 Annoyance

Figure 4.18 shows ratings of annoyance. Task had a main effect ($F(2,22)=7.06$, $p<0.004$). Post hoc tests showed that Route Planning ($\mu=2.43$) caused users to experience more Annoyance than Document Editing ($\mu=1.6$, $p<0.025$) and Email Classification ($\mu=1.5$, $p<0.004$). No other differences were found. The results are mostly consistent with the ratings of subjective workload, as the highest workload task caused the most annoyance and the lowest workload task caused the least annoyance.

Timing had a main effect on ratings of annoyance ($F(3,33)=7.41$, $p<0.001$). Post hoc tests showed that interruptions at Best moments ($\mu=1.69$) caused 18% less annoyance than at Worst moments ($\mu=2.07$, $p<0.079$) and 28% less annoyance than at Random moments ($\mu=2.35$, $p<0.052$). Not surprisingly, users experienced the least annoyance when not interrupted ($\mu=1.26$) than when interrupted ($p<0.043$ in all cases). No other differences were found and there were no interactions.

4.4.7.4 Respect

Figure 4.19 shows the ratings of respect attributed to the interrupting system. Task had no main effect ($F(2,22)=0.70$, $p<0.51$), while Timing did have a main effect on user ratings ($F(3,33)=12.11$, $p<0.001$). Post hoc tests showed that when interrupted at Best moments ($\mu=3.02$), users rated the system to be 63% more respectful to their primary task than when interrupted at Worst moments ($\mu=1.85$, $p<0.015$) and 39% more respectful than when interrupted at Random moments ($\mu=2.17$, $p<0.086$). Users rated the system most respectful to their primary task when not interrupted ($\mu=4.02$) than when it was interrupted ($p<0.001$). No interactions were detected in the data.

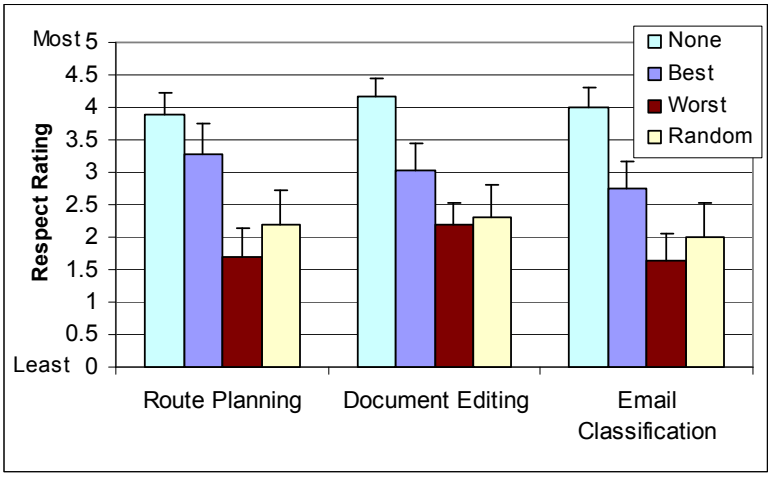


Figure 4.19: Ratings of Respect

4.4.8 Discussion

The moment at which an ongoing primary task is interrupted influences the disruptive effects caused by the interruption. Results showed that interrupting at the Best moments was less disruptive across tasks. Users resumed primary tasks 69% faster, experienced 18% less annoyance, and attributed 63% more respect to the interrupting system compared to being interrupted at the Worst moments. Similar differences were found between Best and Random. However, interrupting at the Worst moments demonstrated no measurable improvement over Random moments, which shows that not all breakpoints are opportune for interruption, especially those that are lower in a task model.

For each task, Best and Worst moments were selected by identifying breakpoints with the lowest and highest workload in the workload-aligned models. Since interrupting primary tasks at Best moments consistently caused less negative impact, mental workload can and should be leveraged to help predict how opportune various moments in a task are for interruption. Also, the average difference in time between the Best and other moments across tasks was relatively short, less than a minute on average (μ_{RP}=57.3s, μ_{DE}=41.1s, μ_{EC}=4.5s), but the corresponding mitigation of negative impact was meaningfully large.

From a more theoretical point of view, the findings imply that the moment at which a notification is delivered relative to a user's ongoing task will affect interruption cost. Indeed, several studies have shown that the moment that a task is interrupted does affect interruption cost, e.g., see (Czerwinski, Cutrell et al. 2000; Monk, Boehm-Davis et al. 2002; Bailey and Konstan 2006; Iqbal and Bailey 2006). In these studies, cost was measured in terms of the time needed to resume the primary task. Our results strongly suggest that the interruptions resulting in lower cost likely occurred at moments of lower workload, when more attentional resources were available for the interrupting task and fewer resources were needed to resume the previously suspended task (Rubinstein, Meyer et al. 2001).

Apart from breakpoints, there may have been other moments of low mental workload during the tasks, which may have been opportune for interruption, but were not considered. We focused on selecting opportune moments from breakpoints since systems could feasibly automate their detection and this process could generalize to many goal-directed tasks. While considering only breakpoints may not always produce the optimal solution, it can still meaningfully mitigate effects of interruption, as shown by our results.

4.5 Implications for Interruption Management Systems

Our goal in this chapter was to empirically understand why interrupting at breakpoints typically results in reduced interruption costs. We performed a study to better facilitate this understanding, exploring the relationship among workload and breakpoints using pupil dilation as a measure of workload. Results from the study validated prior theoretical postulations that users experience temporary reductions in processing efforts (or workload) at breakpoints during interactive task execution. We also made the novel discovery that workload varies across breakpoints existing at different levels of the task hierarchy. In a subsequent study we validated that interrupting at breakpoints with lower workload results in lower interruption cost compared to breakpoints with higher workload, and both have lower interruption costs than random interruptions.

Our findings show that breakpoints have lower cost of interruption likely because of the temporal reductions in processing efforts (workload) at these moments. Furthermore, we showed that workload is lower for breakpoints higher in the task hierarchy and is higher

for breakpoints lower in the hierarchy. For interruption management systems, this has important implications. The goal of interruption management is to find a better balance between the costs of interrupting and the timeliness of the information being conveyed. For example, being able to identify moments during task execution with different levels of interruption cost provides opportunities to better match costs with different levels of notification urgencies.

Our findings could be realized within methods for predicting costs in two ways. The first approach follows directly from the methodology described in this chapter. For example, workload-aligned task models would be developed by aligning a continuous measure of workload to the corresponding models of task execution. From these workload-aligned task models, the breakpoints and subtasks would be rank ordered based on their workload and then mapped to a cost value. The model of the task and associated cost information would then be formally described using a task specification language such as that presented in (Bailey, Adamczyk et al. 2006). As a user performs tasks, a monitoring system would match the ongoing interaction to the specifications, allowing the monitor to identify when a specific breakpoint or other moment was reached. The associated cost value could then be extracted from the specification and directly used to determine whether to interrupt, or passed to a broader reasoning framework (Horvitz, Koch et al. 2004). This approach would be most appropriate for safety critical or other domains where tasks have fairly prescribed sequences and the range of possible tasks is somewhat constrained, but the cost of poorly timed interruption could cause loss of life or catastrophic damage. Example situations might include working through aviation checklists (Degani and Wiener 1993) or entering target information in a command/control interface (Guerlain and Willis 2001).

In other situations where high precision is not necessary, an alternative is to utilize a set of workload-based heuristics to assigns costs within static specifications of tasks. For example, lower costs could be assigned to breakpoints higher in the task structure and successively higher costs could be assigned to breakpoints lower in the structure. Similar heuristics could be developed for different types of subtasks such as memory store and recall, language comprehension and generation, and reasoning. Though applying heuristics could only offer approximations, they could be expediently applied to many

tasks, the values would still allow systems to better reason about when to interrupt, and the use of heuristics would eliminate the need to develop workload-aligned task models. We believe that this is the more appropriate method for most tasks in the desktop domain, which is the domain we are primarily focused on in this thesis.

This thesis seeks to further advance this particular approach. The next step is to determine the workload-based factors that can contribute to a set of heuristics for predicting interruption costs. Our findings in this chapter show *level* of breakpoint to be a reasonable indicator of cost, but there are exceptions where breakpoints on the same levels may have different costs. This suggests that other characteristics of the task structure may also contribute to the demands of the task at certain moments, consequently affecting interruption costs.

Chapter 5

Leveraging Characteristics of Task Structure to Predict Costs of Interruption at Breakpoints

In the previous chapter, we demonstrated that users experience transitory reductions in workload at breakpoints, reductions were larger for breakpoints higher in the task hierarchy, and interrupting at breakpoints results in lower costs. To realize these theoretical findings in practical applications, the next step is to develop methods to predict interruption costs at breakpoints. These methods can be used by interruption reasoning systems to determine when the best moment is to schedule a notification.

One way of identifying lower cost breakpoints is to align workload data to hierarchical task models a priori, rank order breakpoints based on workload and pass this data to a reasoning system. However, using physiological measures of workload (e.g., pupil dilation) is invasive for the user and requires considerable effort. This effort may be justified in determining the cost for tasks in safety critical domains and where precision is important. But for less critical tasks, e.g. desktop tasks, using physiological measures as predictors of interruption cost may not be feasible, since the effort often completely outweighs the perceived benefits.

An alternative is to use characteristics of the task structure to predict cost of interruption. For example, as shown in the previous chapter, *level* (depth) of the breakpoint in the task hierarchy was found to be correlated with the cost, where breakpoints at shallower levels in the hierarchy had lower cost and vice versa. This chapter investigates which additional characteristics of the task structure, if any, could be used to predict cost of interruption. Consideration of additional factors should provide

more accurate costs than simply using level as a predictor, but with much less effort than using physiological measures such as pupil dilation.

We present a study conducted to develop models for predicting cost of interruption at breakpoints. In a subsequent study we test the performance of the model by applying it on a new set of tasks. We conclude this chapter with a discussion on how these models can be used within methods for detecting and differentiating breakpoints.

5.1 Overview

The goal of the study was to determine how well characteristics of task structures could be used predict costs of interruptions (COI) at breakpoints. Characteristics include depth of decomposition, types of subtasks, mental carryover, etc. For costs of interruption (COI), we are primarily focused on *Resumption Lag*, the time it takes to resume a suspended task after addressing an interruption. Resumption lag has been used as a cost metric in many prior efforts (Trafton, Altmann et al. 2003; Altmann and Trafton 2004; Bailey and Konstan 2006), and represents a direct, observable effect of being interrupted.

We first collected COI data from users performing a set of representative tasks in experimental settings. We determined candidate characteristics that may be predictive of the cost, leveraging prior research and our experience from the workload analysis described in Chapter 4. We then identified predictive factors within the candidate characteristic set and determined the number of cost classes to be modeled. Finally, we then applied machine learning techniques to develop a statistical model that mapped the predictive factors to the COI classes. The model was evaluated on a completely new set of interactive tasks, providing insight into the model's generalizability.

5.2 Collection of COI data

The first step in developing the COI model was to collect a sample of resumption lag data. This was achieved by having users perform primary tasks, interrupting the tasks at sample breakpoints with peripheral tasks, and measuring the resumption lag.

5.2.1 Users

12 users (7 female) participated in the study. Users ranged from 23 to 33 years of age ($\underline{M}$=26.33, $\underline{SD}$=2.839). Users were compensated with a $5 coupon to a local coffee shop.

5.2.2 Primary Tasks and Models

Three categories of primary tasks were developed:

Video Editing. As shown in Figure 5.1, users were asked to use Windows MovieMaker to compose a short (~ 1 min) digital video from provided clips, each about 15-20s. Themes of clips included Disney parade, animal antics, soccer highlights, baby bloopers and bicycle stunts. The user reviewed the clips, added a subset of the clips to the editing timeline and edited length/content as desired. Any visual transitions of their choice were then added between clips. Next, the user reviewed provided audio tracks and added the desired track to the video, and then compiled and saved the final video. Users were encouraged to be as creative as possible while still following instructions.

Route Planning. An interactive map with two routes connecting two cities was displayed along with two tables. Each route had three segments and each segment had distance and fare data associated with it, displayed in a tooltip balloon. For the task, the user moved the cursor over a route segment, retrieved the distance and fare data, entered it into the corresponding row in the table, and repeated for the other two segments. The distance and fare columns were then mentally added and the result was entered into the last row. The user repeated this process for the second route and table and then selected the shorter and the cheaper routes from drop down lists. This task is the same task used for the workload analysis in the previous chapter.

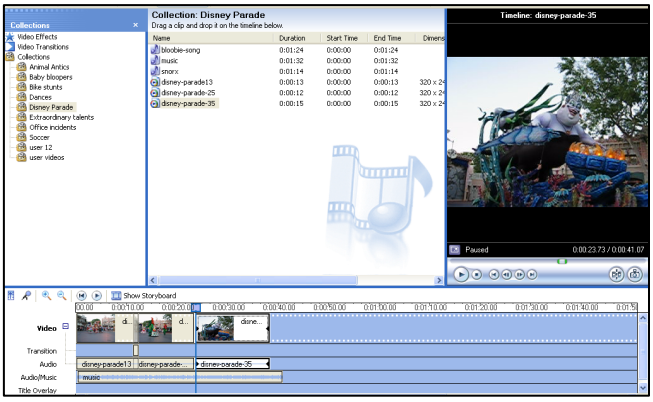


Figure 5.1: The video editing task. A user creates a short digital video by composing and editing provided clips, inserting transitions between clips, and adding a suitable audio track.

Document Editing. Users edited a manuscript annotated with three comments that varied in the complexity of the edit required. Content included contemporary topics such as global warming, legal issues regarding digital media, endangered species, the education system in the U.S., etc. We felt these topics would be interesting and familiar to most users. The user edited the text document according to each comment, stored in a tooltip. After reading a comment, the user located the text, made the appropriate edit, and repeated this process twice more. Once edited, the user saved the document with a name of their choice. This task is also same as the document editing task used for the workload analysis task in the previous chapter.

These tasks were designed to be engaging as well as to have meaningful subtasks requiring varying mental effort, salient breakpoints between the subtasks, and largely prescribed execution sequences. The latter constraint was necessary to be able to interrupt task execution at specific points for data collection. Each task lasted about 5-6 minutes. Since a within-subjects design was used, multiple instances of each task were created and we were careful to alter just the content, not the basic execution structure of the tasks. For example, video editing used different video and audio clips, document editing used different content, and route planning used different city names and route data.

To define the structural characteristics of the tasks, GOMS models were developed, one per category (see Figure 5.2 for the task model for video editing, see the previous chapter for task models for Route Planning and Document Editing). Following (Card, Moran et al. 1983), initial models were built based on our own understanding of the task's execution. The models were iteratively refined by having users (in a pilot study) perform the tasks and matching the models to the observed execution sequences. This continued until the models achieved high accuracy.

In developing the task models, we tried to balance having enough detail to identify lower-level breakpoints with being able to allow for variability in the execution sequences. For example, the model in Figure 5.2 includes a subtask for insert transition at level 3, but whether a user drags or copies and pastes a transition to the timeline is not explicitly modeled. This allowed us to model the adjacent breakpoints yet still capture

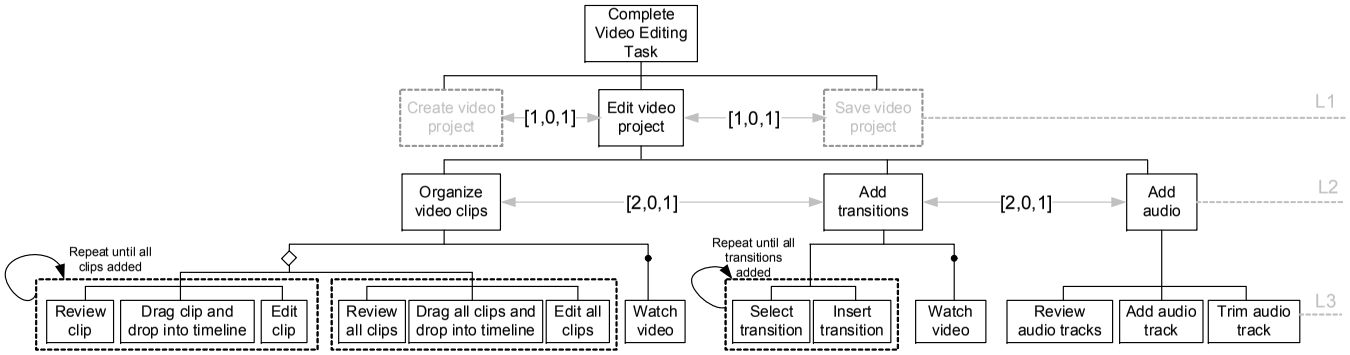


Figure 5.2: Part of the GOMS model for the video editing task, showing details for the Edit Video subtask. Time moves from left to right. The models were developed to strike a balance between having an appropriate level of detail and allowing for variability in execution sequences. Diamonds indicate common alternative sequences that were explicitly modeled and solid dots indicate optional subtasks. Values for predictors ([*level, carryover, difficulty of next subtask*]) are shown for the first two levels of breakpoints in the model.

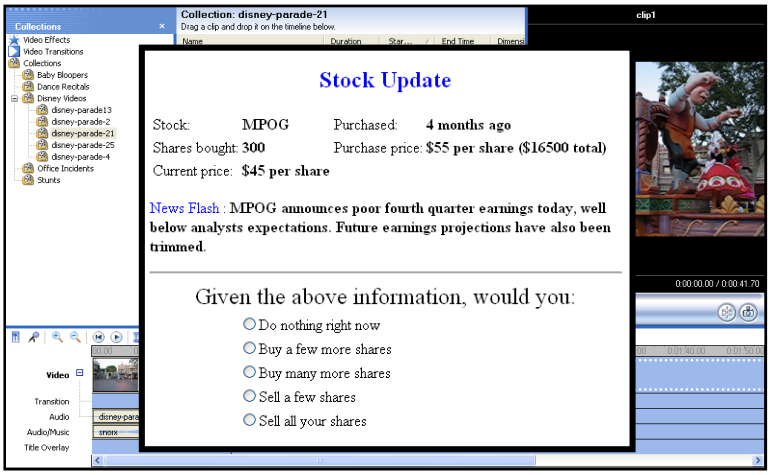


Figure 5.3: Example of the modal window presenting the stock scenario. The window occluded the ongoing task view, forcing the user to switch to the stock task.

some variability. We found that decomposing a task to about 3-5 levels typically achieved the desired balance.

The final models were evaluated against the interaction sequences from the actual study. Each model achieved more than 90% accuracy with no obvious patterns in the errors.

5.2.3 Peripheral Tasks

A realistic stock scenario was presented as a modal window occluding the application in focus. The stock task was adapted from prior work (Bailey and Konstan 2005; Bailey and Konstan 2006). Each scenario consisted of a fictitious company's name along with the quantity, date, and price of shares that the user hypothetically purchased from that company (see Figure 5.3). Each scenario also contained the price of the stock and a one sentence "news-flash" about the company. After analyzing the scenario, the user selected one of five trading actions; do nothing, buy a few more shares, buy many more shares, sell a few shares or sell all shares. Multiple instances of the task were created and each required about 20s to perform.

This task was used because it is representative of peripheral information that users often receive (Maglio and Campbell 2000; McCrickard, Catrambone et al. 2003) and because analyzing the scenarios taps cognitive resources (Wickens 2002).

5.2.4 Moments for Interruption

For each task, we selected a sample of ten representative subtask breakpoints from the corresponding GOMS model. For example, for video editing, breakpoints included the point after dragging a clip and dropping it on the timeline, but before making any edits; after making the last edit on the timeline, but before adding transitions; after completing the video editing but before saving it, etc. The set of selected breakpoints sampled different levels and temporal positions in the task model. Breakpoints for the other tasks were selected using a similar strategy.

5.2.5 Experimental Setup

A Wizard of Oz technique was used to time delivery of the peripheral task. The experimenter monitored a user's task execution using a Real VNC client and delivered peripheral tasks at the selected breakpoints using a remote command.

For each selected breakpoint, the experimenter waited for the user to make a directed action signifying the start of the subsequent subtask, based on the task model. This method mimicked how systems may identify breakpoints in practice (Bailey, Adamczyk et al. 2005). Since a high speed LAN connection was used, there was negligible latency from when the peripheral task was commanded to when it actually appeared on a user's screen.

5.2.6 Procedure

Upon arrival at the lab, a user went through an informed consent process and received general instructions for the study. Since a within-subjects design was used, the primary task categories were presented using a Latin Square design.

For each category, the user received specific instructions, performed a practice and then performed the actual trials. A user performed five task trials and was interrupted twice during each trial. Interruption moments were randomly selected from the defined set of ten, without replacement. The peripheral task was presented in a modal window and covered the main work area, prompting a task switch at the defined moment. Users were asked to begin the peripheral task as soon as it appeared and, once complete, to resume the primary task as quickly as possible. This process was repeated for each task category. The study lasted 90 min.

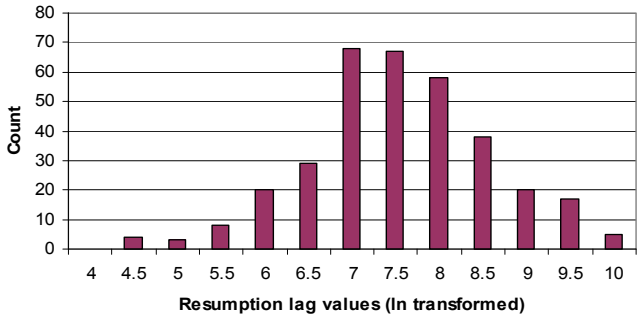


Figure 5.4: Histogram of the transformed resumption lag data. The distribution is near normal, with more values in the middle and fewer at either end of the scale. Each bar represents the number of values that fall between the value corresponding to the previous bar and itself.

5.2.7 Measurements

Resumption lag was used as the cost of interruption. It was measured as the time difference from when the peripheral task window was closed to when the user made the first observable action in the resumed primary task (Altmann and Trafton 2004).

Other measures could have included error rate and affective state. Errors were not used because they would be difficult to judge for creative tasks such as video editing and it is unclear what time window should be used for attributing errors to the interruptions. Affective state was not used since this is often measured using a subjective rating, which would likely change based on the interruption's content. Resumption lag is objective, continuous, and well defined.

5.2.8 Results

A total of 360 resumption lag values were collected. To normalize the resumption lag data, a natural log transform was applied, which is common for performance data. Outliers and data values corresponding to errors (e.g., a breakpoint was missed due to the user deviating from the model) were removed from the data (~ 6%). This left a total of 337 samples in the data set. As shown in Figure 5.4, the resulting transformed data set ranged from 4.10 to 9.86 ($\underline{M}$=7.3, $\underline{SD}$=1.04). These results are consistent with resumption lag data reported in prior work (Trafton, Altmann et al. 2003; Altmann and Trafton 2004; Iqbal and Bailey 2005).

With this transformed data, we proceeded to building the COI model. This consisted of identifying candidate factors, using stepwise regression to determine the most predictive ones, clustering the data into discrete classes, and learning a mapping from the predictive factors to those classes.

5.3 Identification of Candidate Factors

The next step was to propose a candidate set of structural characteristics related to breakpoints. Based on prior work and our own experience, we identified these factors:

Level. The level of a subtask breakpoint is defined as (1 +) the depth of the shared ancestor of the adjacent subtasks. This factor was selected based on our prior analysis showing breakpoints to have lower interruption costs and breakpoints at coarser levels having even lower cost than breakpoints at finer levels.

Presence of a visual resumption cue. This factor was a binary value (0=no cue) indicating whether the state of the primary task at the point of interruption provides an obvious visual cue for resuming it. The *presence* (not saliency) of cues has been thought to reduce COI (Altmann and Trafton 2004; Chung and Byrne 2004).

Percent of task complete. This refers to how much of the overall task is complete at the breakpoint. To provide an accurate value, we timed a few users performing the tasks and then mapped the percent complete to each breakpoint in the model. Temporal position in a task (e.g., beginning, middle, end) has been shown to affect COI (Miyata and Norman 1986; Czerwinski, Cutrell et al. 2000; Monk, Boehm-Davis et al. 2002).

Percent of parent subtask complete. This factor was similar to the previous one, except that percent complete was now measured relative to the parent of the subtasks adjacent to the breakpoint. This factor was considered because users often chunk execution of tasks (Navon and Gopher 1979).

Difficulty of preceding subtask. There is no standard method for computing the difficulty of *subtasks*. We thus adapted a heuristic often used to approximate difficulty when predicting resource conflicts between tasks (Wickens 2002). The leaf subtasks (operators) were categorized based on presumed difficulty and the categories were qualitatively ordered based on their presumed cognitive demands. As shown in Table 1, this produced

Difficulty	Category	Example
1 (Least)	Motor movements	Move mouse towards a menu item or select a menu item
2	Routine content generation	Enter a new filename or select a transition for a video clip
3	Comprehension or store information in memory	Read text or comments, retrieve a route segment's distance and fare information and commit it to memory
4	Recall information	Recall a route segment's distance and fare information
5	Creative content generation	Edit document text or edit video clips
6 (Most)	Mathematical reasoning	Add distance or fare information

Table 5.1: The six levels of subtask difficulty in our tasks, their corresponding categories, and examples of each.

6 categories, with '1' being least demanding. For example, a mouse movement was assigned 1 whereas mental calculation was assigned 6. If the preceding subtask was a goal subtask, the difficulty of its last operator was used. For example, the breakpoint between Edit video project and Save video project in the video editing task was assigned 1, as this was the difficulty of the last operator (Trim audio track). Difficulty of preceding subtask was considered because COI is thought to depend on a user's mental workload at the point of interruption (Bailey and Konstan 2005).

Difficulty of next subtask. This factor was included for the same reason as the previous one and its value was computed analogously. If the next subtask was a goal subtask, the difficulty of its first operator was used.

Carry over at breakpoints. This factor refers to how much data must be maintained across a breakpoint. Similar to difficulty, we categorized breakpoints based on presumed carryover, resulting in four categories, and qualitatively ordered them by assigning values of 0 (no carryover) to 3 (high carryover). For example, maintaining a seven digit value across a breakpoint in route planning was assigned 3, while retaining where to position a clip in a video after selection was assigned 1. We included this factor since it provided another estimate of workload at a breakpoint.

These values were computed for each breakpoint in the task models. Though additional characteristics could have been included, we restricted this first set to those identified in prior work and that could be computed relatively easily.

5.4 Determination of the Most Predictive Factors

The next step was to determine which of the candidate factors were the most predictive of resumption lag. We used a stepwise multiple regression for this purpose. We first checked the global utility of the regression model. A multiple regression analysis with Resumption Lag as the dependent variable and all candidate factors as independent variables was performed. The linear regression model was predictive ($F(12,336)=11.23$, $p<0.0001$, adjusted $R^2=0.25$) and the residuals of the regression model met the normality

Model	*β*	Std Err	*t*	*p*
Constant	6.197	0.138	44.92	0.0001
Level	0.38	0.068	5.581	0.0001
Carry Over	0.158	0.067	2.351	0.019
Difficulty	0.077	0.038	2.063	0.040

Table 5.2: Regression model with the three predictive factors.

assumption. Passing the global utility test strongly suggests that at least one of the candidate factors has a non-zero coefficient and is predictive of resumption lag.

To create a parsimonious model (the least number of factors that explain as much of the variance in the data as possible), a stepwise model building technique was employed. As summarized in Table 5.2, this technique showed that Level, Carry Over, and Difficulty of Next Subtask were the most predictive of Resumption Lag, with adjusted $R^2 = 0.26$. This means that 26% of the variance in Resumption Lag can be explained by these three characteristics alone, a promising result given the innate complexities of the human information processing system (Card, Moran et al. 1983).

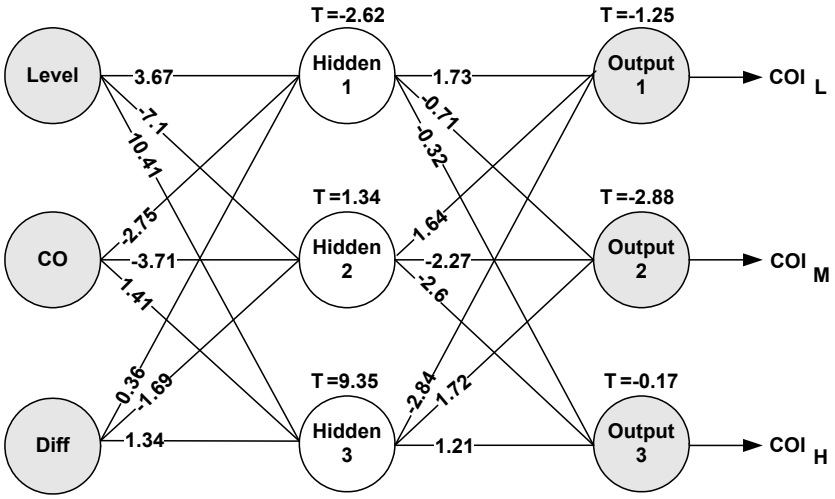


Figure 5.5: The MLP model that maps the predictors (Level, Carryover, and Difficulty of next subtask) to the COI classes. The input nodes are the predictors identified from the stepwise regression analysis while the output nodes correspond to the COI classes determined from the regression analysis.

When compared to the full model, the amount of variance explained changed little, yet the number of factors was reduced to three. Also, the reduction to these three particular factors suggests that COI depends more on the characteristics that reflect current and prospective allocation of mental resources (workload) than on those that reflect temporal position or cue availability in the task.

An interruption reasoning system could use this model to predict resumption lag values for each breakpoint in a task model. However, given the model's modest correlation coefficient, a challenge for systems is to interpret the meaningfulness of differences among predicted values (e.g., how much better is 7 than 7.5?), as each prediction has error associated with it. Though a z-score accounts for variance in the data set, it does not account for the error in the predicted value itself.

Thus, we decided to cluster resumption lag into classes such that there was a meaningful difference between them. This would allow the model to be adapted to predict the *classes* rather than specific values, which would enable increased prediction accuracy, at the price of decreased sensitivity.

5.5 Determine Cost Classes

To determine the number of cost classes the data could be classified into, we applied K-means cluster analysis to the data. The goal was to identify the largest number of clusters such that meaningful differences would be maintained between them.

Based on several data visualization techniques, we found that about 3-5 clusters would be appropriate. Analyzing each number, we found the use of 3 clusters (COI_L, COI_M, and COI_H) to be most appropriate. With these clusters, 75 values fell into COI_L ($\underline{M}$=5.938, $\underline{SD}$=0.612), 177 values into COI_M ($\underline{M}$=7.252, $\underline{SD}$=0.376) and 85 values into COI_H ($\underline{M}$=8.628, $\underline{SD}$=0.505). An ANOVA showed that the means differed ($p<.0001$ between all pairs). Three clusters were the most that maintained these differences between pairs.

This is consistent with (Fogarty, Ko et al. 2005), where it was reported that a user's interruptibility could be best classified into at most 3 classes. Our result is an interesting parallel, as it suggests that, absent a physiological measure, a system may only be able to effectively classify the COI into at most 3 classes.

5.6 Learn a Mapping from Predictors to COI Classes

Finally, we needed a mechanism to map from the predictors to these COI classes. Unfortunately, the regression equation could not be used since the constant term (6.197) was greater than the mean of COI_L (5.938), thus it could not always map predictors to this class. After analyzing several methods, we settled on the use of a multi-layer perceptron (MLP). Unlike a Naïve Bayes model, for example, an MLP model does not require the predictors to be independent.

		Predicted Cost			
		COI_L	**COI_M**	**COI_H**	**Total**
Actual Cost	**COI_L**	42 (56%)	32(42.7%)	1(1.3%)	75 (100%)
	COI_M	24 (13.6%)	136 (76.8%)	17(9.6%)	177 (100%)
	COI_H	4 (4.7%)	46 (54.1%)	35(41.2%)	85 (100%)

Table 5.3: Distribution of predicted vs. actual COI classes for the model building tasks.

Back propagation was used to learn an MLP model, with Level, Carry Over and Difficulty of Next Subtask as input. There was one hidden layer and three outputs, one for each COI class. Figure 5.5 shows the resulting MLP.

A 10-fold cross validation technique was used to evaluate the model. Table 5.3 shows the distribution of predicted vs. actual COI classes, where the diagonal represents correct predictions. Total number of correct predictions was 63.2%, much better than chance ($N(0.33, .00066)=24.67$, $p<.0001$).

A two-way contingency table analysis shows that the actual cost classes are related to the predicted classes (Pearson $\chi^2(4, N=337)=120.17$, $p<.0001$). Pairwise comparisons showed that the number of correctly predicted COI_L and COI_M classes were greater than those that were incorrectly predicted ($p<0.0001$). The model was slightly less accurate for predicting COI_H, as it tended to predict the adjacent class. However, the most egregious type of error, predicting COI_L when it was actually COI_H, was very low (4.7%).

The next step was to evaluate how well the model predicted COI classes when applied to breakpoints within tasks that are *different* from those used in the model building process.

5.7 Evaluating the COI Model on Novel Tasks

We conducted a second experiment to evaluate how well our COI model (the MLP plus heuristics for assigning its inputs) predicted COI classes when applied to breakpoints within different primary tasks. Specifically, we wanted to (i) evaluate the accuracy of the predicted COI classes and (ii) test whether there are differences in resumption lag between predicted COI classes, which would validate that reasoning systems should integrate the use of our model.

5.7.1 Users

A different set of 12 users (2 female) participated in the study, with ages from 21 to 26 ($\underline{M}=24.2$, $\underline{SD}=1.8$). Users were compensated with a $5 coupon to a local coffee shop.

5.7.2 Primary Tasks and Models

Two new primary tasks were developed for this experiment:

Collage Generation. As shown in Figure 5.6, users were asked to create collages in Adobe Photoshop that would communicate a given theme. Themes included activities in amusement parks, life as a CS student, and experiences in summer camps. To foster engagement in the task, users were told that the collages would be used for marketing. For each theme, four categories, each with four images, were provided (e.g., outreach, research, campus, and fun for *life as a CS student*). To create the collage, users created a blank image with a specified width and height, and then opened all of the source images (16 in total) that could be used in the collage. Users were to select at least one image

Figure 5.6: Collage generation task. A user created a collage by composing images from several categories depicting a certain theme. Users included at least one image from each category, manipulated the layers, and add visual effects to the collage

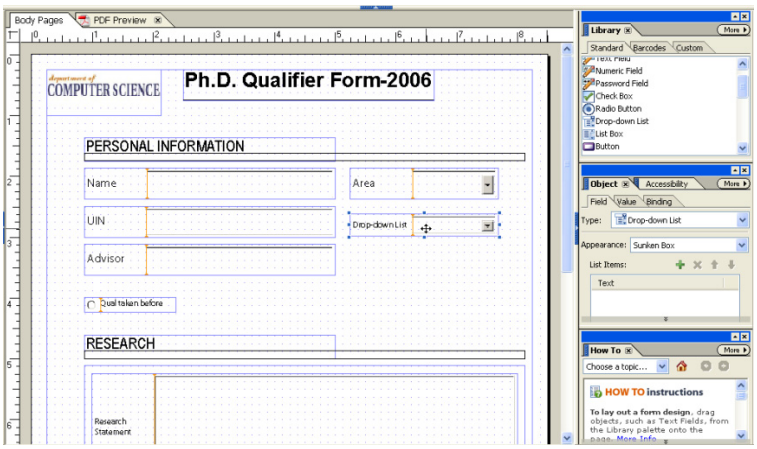


Figure 5.7: The form design task. A user creates an electronic form by dragging appropriate widgets from the library on the right and placing them on the form. Widgets are parameterized according to the requirements of the task.

from each category, paste it into the collage, and size and position it as necessary. After integrating the images, users manipulated their layer ordering to create the desired look and added at least one visual effect (e.g., blurred edges around each layer) to the collage. Finally, they saved the collage in a directory with a desired name.

Electronic Form Design. Users were asked to design electronic forms using Adobe Designer (see Figure 5.7). Forms included a registration form for an HCI workshop, customer satisfaction survey, purchase order, and Ph.D. qualifying exam. Users were provided with a partially completed form and asked to complete it based on given requirements. For example, users were asked to construct fields for collecting payment information, feedback about customer service, and product information as efficiently as possible. Widgets such as text fields, check boxes, radio buttons, and drop-down lists were available for use. Users selected any widgets they felt were most suitable for collecting the needed information, placed them on the form, and added the appropriate text.

Similar to the tasks used in the first experiment, these tasks were designed to be

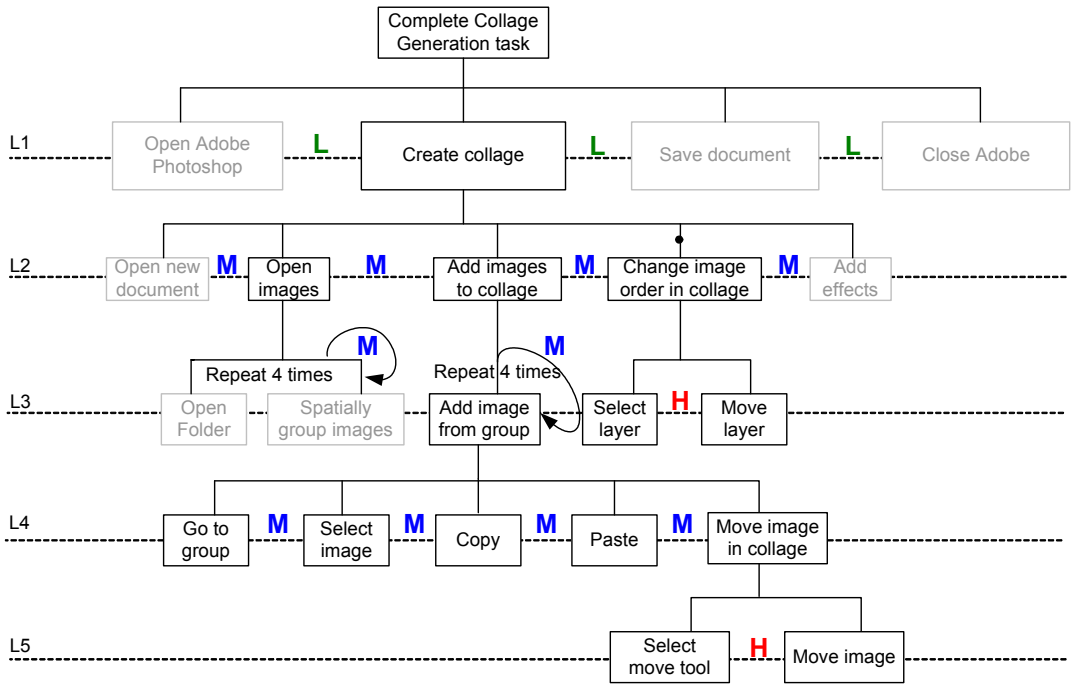


Figure 5.8: Part of the GOMS model for the collage generation task, showing details for just the Create collage subtask. The predicted COI classes are shown at each subtask breakpoint.

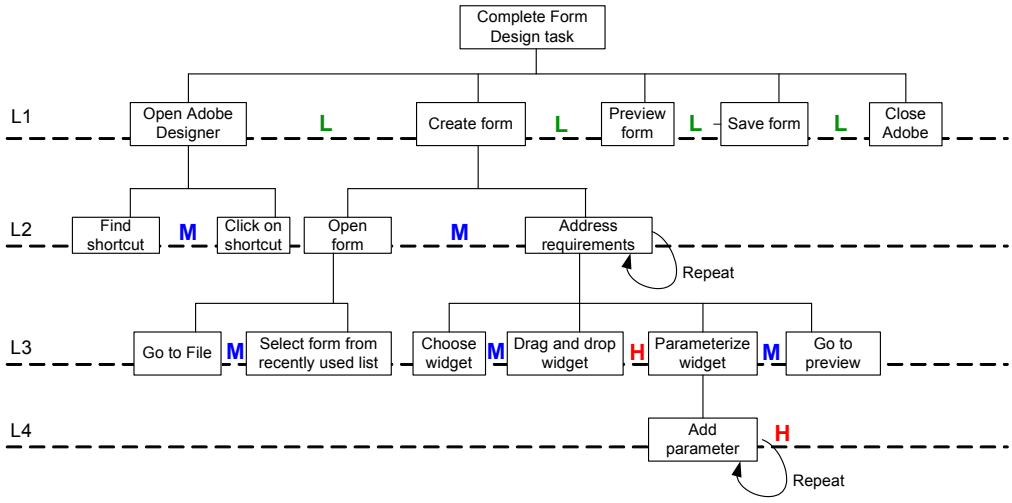


Figure 5.9: GOMS model for the form design task. Predicted classes are shown at each breakpoint.

engaging and to have subtasks requiring varying mental effort, observable breakpoints, and mostly prescribed sequences. Each task took about 5-6 minutes to perform. Task models were iteratively developed and validated using techniques as before. Error rates of the task models were low, consistent with the first experiment.

5.7.3 Predicted COI Classes and Moments for Interruption

We applied our COI model, consisting of the heuristics for assigning values to the predictors and the MLP shown in Figure 5.5, to predict the COI class at each subtask breakpoint. For each breakpoint, we used our heuristics to assign values for the three predictors (Level, Carry Over, and Difficulty of Next Subtask). These values were then used as input to the MLP, which computed the predicted COI class.

Figure 5.8 and Figure 5.9 show part of the task models for collage generation and form design respectively, with the predicted COI classes. Including both task models, there was a total of 38 subtask breakpoints, of which 7 were assigned to COI_L, 26 to COI_M, and 5 to COI_H.

For the specific moments to interrupt, we randomly selected a sample of six breakpoints from each task model, two from each of the three COI classes. The peripheral task, experimental setup and procedure, and resumption lag measurements were the same as in the first experiment.

5.8 Results of Model Evaluation

A total of 144 data samples were collected. Prior to analysis, we filtered outliers and any data resulting from experimental errors, resulting in 7% of the values being removed. This left 134 samples in the data set. Once filtered, a natural log transform was applied to normalize the resumption lag data.

5.8.1 Compare Predicted to Actual COI Classes

The resumption lag values at each breakpoint were classified into their *actual* COI classes using the cluster information determined in Experiment 1.

Table 5.4 shows the distribution of predicted vs. actual COI classes. Actual cost classes are related to the predicted cost classes (Pearson $\chi^2(4, N=134)=39.96$, $p<.0001$). Follow-up pairwise comparisons showed that the number of correctly predicted classes (56% for COL_L, 49% for COL_M, and 54% for COL_H) were significantly greater than those that were incorrectly predicted ($p<0.0001$). Overall, our model correctly predicted 53% of COI values, much better than chance ($N(0.33, 0.00165)=13.05$, $p<0.0001$).

		Predicted Cost			
		COI_L	**COI_M**	**COI_H**	Total
Actual Cost	**COI_L**	35 (55.56%)	19 (30.16%)	9(14.29%)	63 (100)%
	COI_M	2 (4.26%)	23(48.94%)	22 (46.81%)	47 (100)%
	COI_H	5 (20.83%)	6 (25%)	13 (54.17%)	24 (100)%

Table 5.4: Distribution of predicted vs. actual COI classes for the primary tasks in the second evaluation.

The most egregious type of error (predicting COI_L when it is actually COI_H) was higher than for the model building tasks (20.8% vs. 4.7%), though it was still reasonably low overall. One plausible explanation is that users may have experienced increased mental workload across higher-level breakpoints in these tasks. This would likely cause greater resumption lag, while the model would predict a lower cost.

The classification accuracy for COI_L (~56%) is identical to what was obtained for the model building tasks, while the classification accuracy for COI_M decreased from 77% to

49%, and accuracy for COI_H increased from 40% to 54%. As before, most errors were made to an adjacent class and fewer were made between COI_L and COI_H.

Though changes in the distribution occurred, they were not unexpected as our heuristics can only approximate values for the predictors of Carryover and Difficulty Next Subtask. The most important outcome, however, was that the overall accuracy and pattern of distribution was very similar to the model building tasks. This suggests that our COI model can be reasonably generalized to other goal-directed tasks.

5.8.2 Differences in Resumption Lag among Predicted COI

For this analysis, we grouped the resumption lag values by their *predicted* (not actual) COI values. An ANOVA showed that resumption lag was different among predicted COI classes ($F(2,131)=25.23$, $p<0.0001$). Post hoc tests showed that COI_H ($M=7.44$, $SD=0.92$) had greater resumption lag than COI_M ($M=6.92$, $SD=0.66$, $p<0.013$) and COI_L ($M=6.14$, $SD=0.97$, $p<0.0001$) and that COI_M had greater resumption lag than COI_L ($p<0.0001$). The means of each predicted COI class translates into 1702ms (COI_H), 1012ms (COI_M), and 464ms (COI_L) respectively. These values represent meaningful differences for resumption lag, especially when extrapolated over many interruptions.

Using predicted COI to group resumption lag values was important, as the results show that even with some errors, the model is accurate enough such that predicted values still correspond to empirical, meaningful differences in the cost of interruption. This validates that a system can and should use our model to differentiate among subtask breakpoints, enabling more effective decisions about when to interrupt.

5.9 Discussion

This research explored how well structural characteristics of a task could be used to differentiate COI among subtask breakpoints. Our work has made several contributions in this direction. First, we drew upon literature in cognitive psychology and our prior work to establish that systems need to consider task structure when reasoning about when to interrupt. Second, using data collected in an experiment, we showed that three characteristics of task structure (Level, Carryover, and Difficulty of Next Subtask) can be used to predict COI at breakpoints with reasonably high accuracy and then developed

a parsimonious model that maps these predictors to a set of discrete COI classes. Third, our model was applied to predict COI at breakpoints within different tasks. Results showed that reasonably high classification accuracy was achieved. Results also showed that predicted COIs corresponded to meaningful differences in resumption lag, validating that systems can and should use our model to differentiate among subtask breakpoints.

Cognitive theory argues that lower COI should result when a primary task is interrupted at moments of lower workload, as fewer mental resources must be re-acquired to resume the task (Wickens 2002). The efficacy of our model thus derives from its ability to capture the current (Level and Carryover) and prospective (Difficulty of Next Subtask) allocation of mental resources (workload) at subtask breakpoints. This finding demonstrates a more direct relationship between task demands and interruption costs.

One caveat of this work is that the presence, location, or utility of breakpoints may change as a user's knowledge of performing a task transitions from novel to skilled behavior. As a task becomes skilled, the mental representations are thought to become coarser (Newell and Rosenbloom 1981), eliminating or reducing the utility of some breakpoints. However, experimental studies have shown that familiarity with a task seems to have little effect on how users perceive its hierarchical structure (Zacks, Tversky et al. 2001), suggesting that the mental representations for tasks remain fairly stable. Still, skilled tasks are typically performed in larger chunks (Tollinger, Lewis et al. 2005) and COI models should consider this effect. A possible solution is to extend our current COI model to include skill level as a predictor and to encode a COI value for each skill level at each breakpoint or other salient point in the task.

In the following sections, we describe how the model can be applied in practice, and also discuss how these (or similar) models can serve more free-form tasks.

5.9.1 Applying the COI Model in Practice

The COI model developed in this part of the research are applicable for those tasks that have mostly prescribed execution sequences. These models can be used within interruption reasoning frameworks that include a language for describing tasks and a system for monitoring execution of those tasks. Description of an example system can be found in (Bailey, Adamczyk et al. 2005). The language allows the structure and execution sequences of a task to be concisely described in machine readable format, but in much

less detail than those used for user simulation (Ritter, Baxter et al. 2000). COI values can be assigned to any point in a description, including breakpoints. To apply our COI model, a person computes values for the predictors (Level, Carryover, Difficulty of Next Subtask) at each breakpoint a priori, inputs the values into the MLP (Figure 5.5), and encodes the COI predictions within the description. During task execution, interface events are matched to the task descriptions. When a user reaches a subtask breakpoint, as indicated in the description, the encoded COI value is retrieved and can be sent to a broader reasoning framework. The framework could then consider this value along with social and environmental cues to determine an overall COI.

The benefit of using our COI model and task framework is that it will enable reasoning systems to ground at least part of their COI prediction in cognitive theories of resource allocation related to task structure, which has not been directly considered in existing systems. This is important since resource allocation strongly influences the cognitive cost of interruption (Wickens 2002) and other types of task switching (Rubinstein, Meyer et al. 2001). By considering this information, systems can make more effective decisions about when to interrupt, mitigating competition for resources and thus the COI. Our current COI model would yield the most benefit if it were applied to high frequency, routine, or safety critical tasks, which often have prescribed execution sequences (Degani and Wiener 1993).

5.9.2 Limitations of Using the Current COI Model

The current COI model assumes a stable goal structure and mostly prescribed execution sequences, as these impact the values of the predictors. This means that our current COI model is best suited for tasks that meet these constraints, e.g., high frequency, routine, or safety critical tasks. One approach for addressing this limitation is to create multiple task models, apply our COI model to them, and adapt or develop tools that can automate much of the process, e.g., (John, Prevas et al. 2004; Dragunov, Dietterich et al. 2005). Also, when simpler tasks are composed into more complex activities, the COI values assigned to the simpler tasks cannot be directly applied to the composition. The current solution requires that the COI values be recalculated by fully applying the COI model to the broader activity.

Applying the process of developing COI models described in this chapter is more difficult for tasks that do not have prescribed execution sequences, i.e., that are more free form in nature. This is an important issue to resolve, since most tasks in the desktop domain seldom follow a prescribed path. One way to address this issue is to create machine parsable descriptions for each variation of a task using the task description language discussed earlier, and assign cost to breakpoints in each description using the COI model. Though this would require a large effort, it may be possible to develop or adapt tools to automate much of the process in the future (e.g., see task modeling tools discussed in (John, Prevas et al. 2004; Dragunov, Dietterich et al. 2005; Tollinger, Lewis et al. 2005).

Even though the use of automated tools can reduce the effort in creating the task descriptions, development of an expressive language for adequately describing tasks is not trivial. While this effort is worthwhile for tasks that are safety critical and require high precision, for more open-ended desktop tasks this is not a very feasible approach. Moreover, most tasks in the desktop are freeform, making it exceedingly difficult to describe as static models using description languages. Also, from the point of view of interruption management, we are only interested in the breakpoints per se, and descriptions of the complete task are unnecessary.

The challenge is to develop a method that can detect and differentiate breakpoints without knowledge of the underlying task. This challenge is addressed in the next chapter.

Chapter 6

Developing Task-Independent Statistical Models for Detecting and Differentiating Breakpoints

In the previous chapter we developed methods for predicting interruption cost at breakpoints by leveraging characteristics of the task structure. This approach necessitates development of an expressive language to create descriptions of tasks that these methods can use. For tasks that are safety critical and require precise cost predictions, the effort to develop these task description languages can be justified. However, for less critical tasks, e.g. regular desktop tasks, expanding the effort is often not feasible. Even if a language is developed, most desktop tasks are freeform (Czerwinski, Horvitz et al. 2004) and often difficult to express in terms of a static specification.

In this chapter we seek to overcome this central limitation by understanding how to detect breakpoints and differentiate their granularity without requiring any task specification. *Granularity* refers to the degree of perceptual difference of the actions surrounding a breakpoint (Zacks, Tversky et al. 2001). For example, a breakpoint at a coarser granularity may be when a user has just finished a coding task and is about to switch to check her email. A breakpoint at a finer granularity may be between finishing compiling code and starting to edit code again. Being able to detect and differentiate breakpoints independent of the underlying tasks would allow systems to reason about whether to defer notifications until coarser breakpoints, which occur less often, but offer larger reductions in cost; or until finer breakpoints, which occur more often, but offer smaller reductions in cost (Bailey and Konstan 2006).

A point to note is that *granularity* is not exactly the same as *level*, as previously discussed. The major difference is that granularity refers to the degree of perceptual

differences between actions, whereas level refers to the hierarchical structure of a task. Although there are instances where breakpoints at the same granularity may not exist at the same level, there are many cases where both are the same. Our prior work investigating workload patterns showed workload and interruption cost to vary across breakpoint levels. For now, we assume that each granularity of breakpoints also corresponds to a different cost.

In this chapter we investigate how these three granularities of breakpoints occur during the execution of free-form tasks and examine the feasibility of building statistical models that can detect and differentiate them. We explore features corresponding to interaction events that may be predictive of breakpoints and develop models mapping the features to breakpoints. We conclude with a discussion on how these models can be used to realize defer-to-breakpoint policies for interruption management in practice.

A basic question is how many granularities of breakpoints are detectable and meaningful during task execution. From studies of event perception (Zacks, Tversky et al. 2001; Zacks and Tversky 2001) and task interruption (Fogarty, Ko et al. 2005; Iqbal and Bailey 2006), there is evidence for at least three perceptually meaningful granularities; *Coarse*, *Medium*, and *Fine*. For example, when editing documents, Fine may be switching paragraphs; Medium may be switching documents; and Coarse may be switching to an activity other than editing.

6.1 Overview of the Model Building Process

To develop effective and efficient models for detecting and differentiating task breakpoints, our process was to:

- Collect representative samples of users' task execution, in the form of screen interaction videos and event logs.
- Have observers review the videos, identify perceived breakpoints and their type, and explain their rationale.
- Select those breakpoints with a high degree of agreement, and use them as the ground truth for building the models.
- Identify features describing the interaction at the selected breakpoints, guided by users' explanations, and compute values for the features based on the videos and logs.

- Learn statistical models that map the predictive features to the ground truth values, and evaluate their accuracy.

To facilitate collection and analysis of the data, we developed several new software tools. Activity Recorder records a user's screen interaction and logs system events; Breakpoint Annotator enables observers to review videos, identify breakpoints, and enter linguistic explanations; and Breakpoint Analyzer supports interactive analysis of the data. Our tools can be used to reduce the effort required to collect and analyze similar data, e.g., data in (Newtson 1973; Newtson, Enquist et al. 1977; Zacks, Tversky et al. 2001).

6.2 Collect Task Execution Data

Task execution data was collected from three general task categories; Document Editing (DE), Image Manipulation (IM) and Programming (P). These categories were selected because they are often performed by many users, comprise diverse subtasks, and require varying engagement. Using several categories would allow better understanding of the similarities and differences among breakpoints across tasks.

For each category, two users (6 total) were recruited and screened to ensure they were experienced in the category selected and would be comfortable having their interaction data viewed by others. Users received $20 for participating.

We wanted to collect samples of users' own personal or work tasks, performed in their own environment, ensuring a high degree of ecological validity. Our recording software was thus installed on users' own machines and they were informed of what data it was recording and how to control it. For example, the software allows recording to be started, paused, or stopped at any time using keyboard shortcuts and shows its current status through an icon in the system tray.

The software was configured to record screen interaction at a low, but adequate frame rate (5 fps) using the Camtasia SDK and logged mouse, keyboard, and other relevant system events using the Windows Hooks API. Users were asked to activate the recording software the next time that they would be primarily focused on performing any task within the relevant category for at least an hour. We emphasized that they should perform the task, with the interleaving of any other tasks, as usual. To avoid recording sensitive data, users were reminded that they could pause/ restart the software at any time. Once at

least an hour of data was recorded (minus any pauses), the user notified the experimenter. The experimenter then collected the data and uninstalled the software from the user's machine.

For task content, for DE, one user was writing a research paper while the other was writing study guides for exams. For IM, one user was touching up personal photos from a recent vacation while the other was developing icons and other graphics for a software application. For P, one user was developing a user interface for a research project while the other was writing source code for a course assignment. The applications used included Microsoft Word, Adobe Photoshop, and Eclipse, respectively. Users did temporarily pause collection of their data, but this was very rare overall.

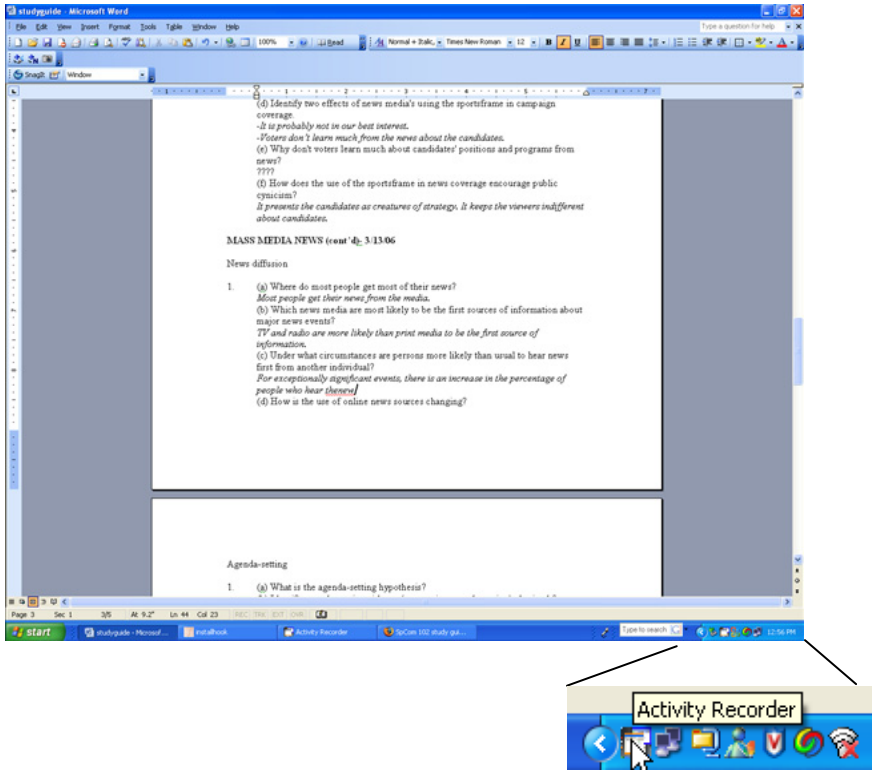


Figure 6.1: Screenshot of data collection during a user's document editing task. The Activity Recorder shown in the system tray executes in the background and collects onscreen activities and system events.

6.3 Identify Perceived Breakpoints and Their Type

The next step was to determine the locations of perceived breakpoints and their type within the task execution data. 24 observers were recruited, 8 per category, and were

asked to review the two videos from an assigned category, mark the location and type of each perceived breakpoint, and enter a brief description as to why they felt this was a breakpoint.

Observers were asked to detect and differentiate three types of breakpoints, guided by the following descriptions:

- *Coarse*. The moment when the largest meaningful and natural unit of execution ends and the next one begins.

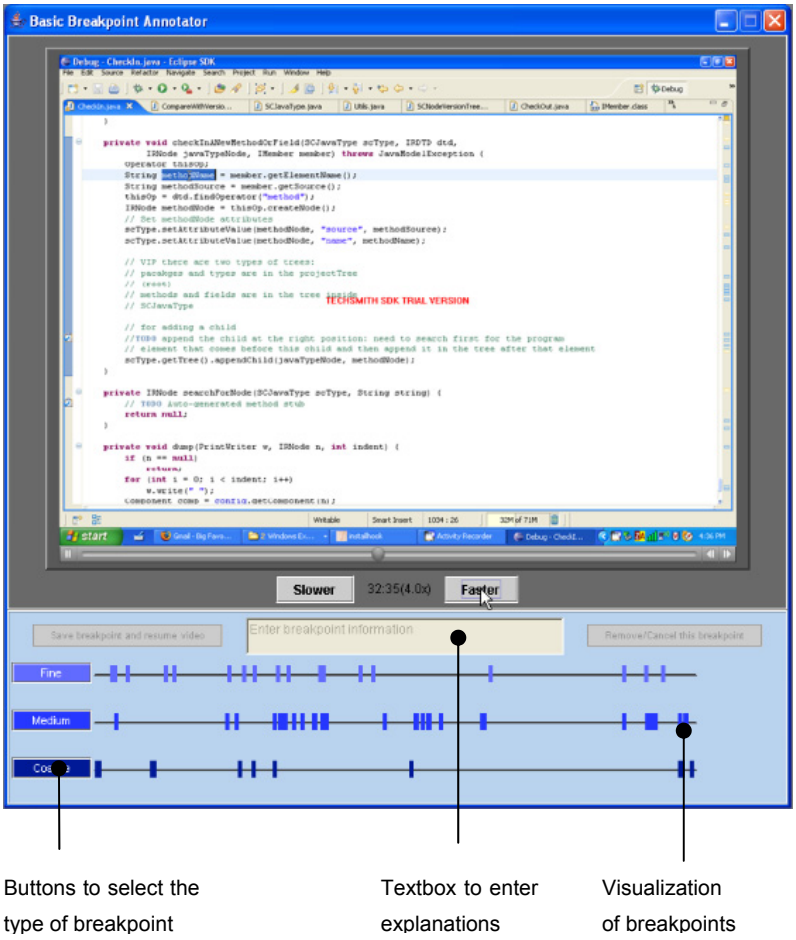


Figure 6.2: Screenshot of the Breakpoint Annotator tool being used to annotate one of the Programming task execution videos.

- *Fine*. The moment when the smallest meaningful and natural unit of execution ends and the next one begins.
- *Medium*. The moment when a natural and meaningful unit of execution, which is smaller than Coarse but larger than Fine, ends and the next one begins.

Inclusion of Coarse and Fine breakpoints, along with their descriptions, is consistent with research on event perception (Newtson 1973; Zacks, Tversky et al. 2001). Medium was included since empirical studies have shown three classes of interruption cost (Fogarty, Ko et al. 2005; Iqbal and Bailey 2006), ostensibly tied to three levels of breakpoints, and results from a pilot study showed that users were able to differentiate the three types of breakpoints within data samples, but not more.

Using observers to identify breakpoints in *another* user's tasks is effective because research has shown that the same schema used to chunk a person's goal-directed actions are also used to chunk their perception when observing another person performing those same actions (Rizzolatti, Fadiga et al. 1996). Also, finding that observers are able to agree on the types and locations of breakpoints would indicate that similar salient cues were being perceived within the interaction data. If those cues could be identified, then models could be built (thinking of models as observers) that automate a similar process.

For procedure, observers came to our lab and were asked to review videos of task execution and identify moments at which they felt that one unit of execution ended and another began; using cursor movements, interaction sequences, and state of the task as cues. The different types of breakpoints were explained using the previous descriptions. The overall methodology was consistent with prior work (Newtson 1973; Newtson and Engquist 1976; Zacks and Tversky 2001).

Our Breakpoint Annotator tool (Figure 6.2) was used to assist the observer in the annotation process. The observer was given a demonstration of the tool and practiced using it on a sample of the data, enabling her to become familiar with the interface and 3 types of breakpoints. Once questions were answered, the observer began annotating the first video.

When a breakpoint was detected, the observer selected a button indicating the type of breakpoint (Coarse, Medium, or Fine). In response, the video was paused, a tick mark was shown on the relevant timeline, and a textbox was activated for entering an explanation. The observer could review the video and modify breakpoints as desired. The observer annotated both videos within an assigned category, but since annotation required about two hours, the process was split across two days. The order of videos in a category was counter-balanced. Observers received $20 for participating.

Category	Task	Coarse	Medium	Fine
Document Editing	DE1	184	226	132
	DE2	140	209	212
Image manipulation	IM1	93	120	293
	IM2	37	99	282
Programming	P1	50	176	193
	P2	252	220	156
Total		756	1050	1268

Table 6.1: Frequency distribution of breakpoints across tasks.

6.4 Summary and Characteristics of Breakpoints

A total of 3074 breakpoints (Coarse=756, Medium=1050, Fine=1268) were identified, and are summarized in Table 6.1. Overall, Fine breakpoints were the most frequent while Coarse breakpoints were the least frequent ($\chi^2(2)=128.9$, $p<0.001$); showing that interactive tasks also tend to be performed in a hierarchical manner (Zacks, Tversky et al. 2001). Interestingly, the distributions for tasks DE1 and P2 show more Coarse and Medium breakpoints than Fine breakpoints. This is not unexpected given users' constant multi-tasking behavior (Czerwinski, Horvitz et al. 2004; Gonzalez and Mark 2004), which, as our results show, may not always be uniform.

Temporal distances between breakpoints are summarized in Table 6.2. The average distance between breakpoints ranges from about 1.5 min (between Fine breakpoints for IM) to 10.7 min (from Fine to Coarse for IM), with the overall average between any two breakpoints being about 3.8 min. These results support and extend data reported in (Gonzalez and Mark 2004; Mark, Gonzalez et al. 2005). Horvitz et al. also report a mean of 43s (s.d. 52s) to transition to an 'available' state from a busy state and show that majority of the busy to available transitions take 1-2 minutes (Horvitz, Apacible et al. 2005).

This data is important because it provides some of the first ecological estimates of how long an interruption reasoning system would need to defer delivery of information in order to reduce interruption cost. For example, assuming that information became available just after a user crossed a Fine breakpoint, delivery of the information would breakpoint), about 4 min to further reduce cost (next Medium), and about 7 min to have minimal cost (next Coarse). These values could also inform the design of interfaces that allow users to specify how long they would be willing to wait for different types of information (Horvitz, Koch et al. 2004).

From observers' explanations, Coarse breakpoints typically corresponded to a switch in high-level activity, indicated by switching to other application(s) judged to be unrelated to the main task, e.g., changing to a music player, checking e-mail, or reading news online. A Coarse breakpoint was also often indicated by returning back to the main application.

Medium breakpoints were tied to switching to applications judged to be relevant to the

	Breakpoint	**Next Coarse**	**Next Medium**	**Next Fine**
Document Editing	Coarse	141 (235)	192 (283)	259 (289)
	Medium	191 (233)	102 (190)	253 (426)
	Fine	259 (367)	175 (356)	112 (231)
Image Manipulation	Coarse	266 (520)	300 (254)	113 (106)
	Medium	538 (564)	244 (330)	117 (167)
	Fine	641 (663)	380 (421)	91 (115)
Programming	Coarse	162 (397)	116 (151)	174 (168)
	Medium	427 (670)	129 (173)	157 (159)
	Fine	402 (623)	142 (157)	139 (186)
Overall Averages	Coarse	190 (365)	203 (239)	182 (219)
	Medium	385 (512)	158 (226)	176 (306)
	Fine	434 (591)	179 (397)	114 (174)

Table 6.2: Mean distances in seconds between adjacent types of breakpoints. Standard deviations are in parenthesis

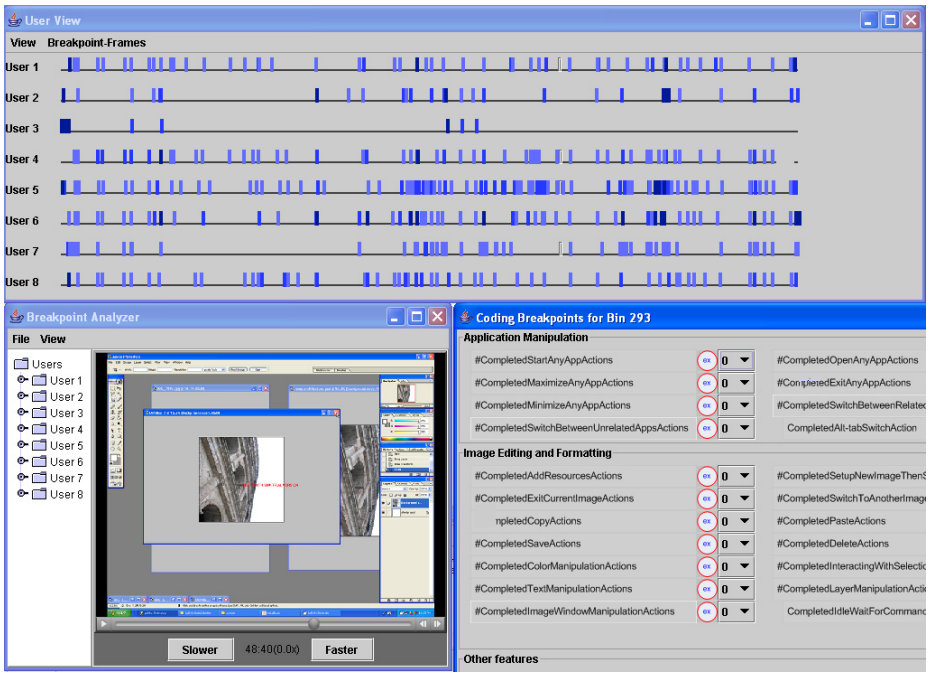


Figure 6.3: A screenshot of our tool that allows breakpoints to be aggregated (top window) and interactively analyzed. When a breakpoint is selected, the video (bottom left) is positioned at the corresponding temporal location. Candidate features are shown at the bottom right window.

primary task or to a large shift in focus within the content of the application. For example, for Document Editing, this included transitioning to edit a paragraph in another section of the document, saving the document, and opening another document. For Image Manipulation, this included loading another image, transitioning to edit a different region or visual feature of the image, and saving the current image. For Programming, this included starting to edit a new class in the file, saving the current source file, switching to another source file, and switching between the code and debug windows.

Fine breakpoints were usually tied to actions on the content within an application. For example, for DE, this included completing formatting commands, searches, and copy/paste sequences; and starting to edit another paragraph near the current insertion point. For IM, this included completing layer manipulations, resize of canvas, and operations such as color adjustments, blending, cropping, and selection. For Programming, this included starting a new method, closing a method, completing a compile, completing the check in/out of a file; and completing definition of class variables.

Interestingly, observers did not identify lower-level units, such as completing a specific sentence or line of code, or moving between fields in a dialog, as Fine breakpoints. The commonly cited reason, clearly evident in the videos, was that editing at the level of a sentence, line of code, or area of pixels exhibited rapid interleaving of pointing, typing, erasing, selecting, scrolling, etc.; thus offering few visually identifiable breaks in the interaction. Thus, attempting to detect breakpoints at this level of detail is probably not warranted, consistent with earlier empirical findings (Iqbal and Bailey 2005).

Overall, this data offers some of the first evidence as to where and how often breakpoints occur within interactive tasks, and offers insight into the types of features that might be useful in models for detecting and differentiating them.

6.5 Identify Ground Truth for Breakpoints

The third step was to combine the breakpoint data across observers and identify breakpoints that had high agreement. This would remove "noise" from the data set and provide the ground truth for the model building process. Figure 6.3 shows a screenshot of our interactive tool that was used to facilitate analysis and coding of the breakpoint data.

We first needed to divide the interaction data into discrete bins, which is necessary since there is natural variance in the temporal locations that refer to the same breakpoint, e.g., some observers may take different amounts of time to decide whether a breakpoint

Category	Task	#Bins	Bins w/ Coarse	Bins w/ Medium	Bins w/ Fine
Document Editing	DE1	360	78	106	85
	DE2	425	60	123	157
Image manipulation	IM1	310	43	85	185
	IM2	434	25	73	160
Programming	P1	406	19	85	126
	P2	371	108	130	110
Total		2306	333	602	823

Table 6.3: Frequency distribution of bins and number of bins with each type of breakpoint. Each bin represents 10s of task execution.

had just occurred.

Our goal was to select a bin size large enough such that slightly different locations referring to the same breakpoint would fall into the same bin, but small enough such that locations referring to different breakpoints would not. Whether a marked location referred to the same breakpoint was determined by analyzing observers' explanations and the corresponding parts of the interaction videos and logs.

From testing a number of bin sizes, between 1s and 20s, we found that a bin size of 10s best met our goal and that this value achieved our goal for each type of breakpoint. This is slightly larger than bin sizes used in prior work (Newtson and Engquist 1976; Hanson and Hirst 1989; Zacks, Tversky et al. 2001), but our tasks were of much longer duration, on the order of hours as opposed to minutes. Table 6.3 shows the number of bins for each task, and how many of those bins contained each type of breakpoint. If a bin had multiple types of breakpoints, it was counted multiple times. A chi-square test showed that observers were biased towards selecting certain bins as breakpoints across all six tasks (DE-1: $\chi^2(359)=1323$, $p<0.0001$; DE-2: $\chi^2(424)=1208$, $p<0.0001$; IM1: $\chi^2(309)=408$, $p<0.0001$; IM2: $\chi^2(433)=997$, $p<0.0001$; P-1: $\chi^2(405)=1183$, $p<0.0001$; P-2: $\chi^2(370)=957$, $p<0.0001$), meaning that the selection of breakpoints was *not* random.

We then had to establish the minimum number of observers who needed to have indicated that a breakpoint was within a bin before being able to conclude that that bin contained a "true" breakpoint. One solution would be to use an absolute threshold (e.g., more than half of the observers must agree), but this does not consider the prior likelihood of agreement.

Category	Task	Coarse	Medium	Fine
Document Editing	DE1	4 (2.3,1.8)	3 (2.1,1.2)	2 (1.5,0.8)
	DE2	4 (2.3,1.3)	3 (1.7,1.0)	2 (1.3,0.7)
Image manipulation	IM1	4 (2.2,1.3)	2 (1.4,0.7)	2 (1.6,0.8)
	IM2	2 (1.5,0.7)	2 (1.3,0.7)	3 (1.7,1.3)
Programming	P1	5 (2.6,1.9)	4 (2.1,1.5)	2 (1.5,0.8)
	P2	4 (2.3,1.8)	3 (1.7,1.0)	2 (1.4,0.8)

Table 6.4: Min number of breakpoints (mean, 1.65*s.d.) that had to be marked within a bin before it was considered a true breakpoint.

Category	Task	Coarse	Medium	Fine
Document Editing	DE1	16 (21%)	40 (38%)	35 (41%)
	DE2	11 (18%)	25 (20%)	38 (24%)
Image manipulation	IM1	8 (19%)	24 (28%)	74 (40%)
	IM2	9 (36%)	19 (26%)	29 (18%)
Programming	P1	40 (21%)	10 (12%)	47 (37%)
	P2	22 (20%)	23 (18%)	11 (10%)

Table 6.5: Distribution of true breakpoints. Percentages indicate what percent of bins (Table 6.3) satisfied the threshold (Table 6.4).

Our approach, following (Hanson and Hirst 1989), was to compute the average number of breakpoints per bin, considering only those bins with at least one breakpoint; add 1.65 standard deviations; and round. This process establishes an alpha=.05 threshold (Hanson and Hirst 1989), and this threshold was calculated for each task and breakpoint type. A bin with a number of breakpoints (same type) greater than the computed threshold was considered to contain a true breakpoint, or *breakpoint bin*. Table 6.4 shows the decision thresholds used in this filtering process.

The number of breakpoints meeting the thresholds was 445 (~25% of all bins with ≥ 1 breakpoint), and are summarized in Table 6.5. Inspection of the table shows that the filtering was fairly uniform. Though this was a stringent filtering process, the aim was to reduce the number of false positives in the data set that would later be used for training.

Also, independent sample t-tests confirm that more observers had detected a breakpoint in a breakpoint bin than in the other bins across tasks and breakpoint type ($p<0.001$ in all cases).

What is perhaps most intriguing about this result is that the observers, all of whom had annotated the videos separately, identified many of the same moments as breakpoints. This occurred because observers were likely perceiving similar cues in the interaction videos. This implies that it should be possible to build models that leverage those same cues to detect and differentiate breakpoints for free-form tasks.

Though there were fewer breakpoint bins due to filtering, the average temporal distances were similar to those listed in Table 6.2 and ranged from 1.4 min to 11.9 min, with the average between any two breakpoint bins being 4.3 min.

6.6 Identify Features Indicating Breakpoints

Next, we needed to identify features that could be used to detect and differentiate breakpoints during task execution. Candidate features were determined based on an analysis of observers' explanations and event logs, our own analysis of the task data, lessons reported in prior work (Fogarty, Hudson et al. 2004; Fogarty, Ko et al. 2005), and whether values could be realistically computed in a system.

For Coarse breakpoints, observers were very consistent in describing them as a switch to another activity that was not related to the main task (and back). However, this abstract description does not yield any specific, usable features and a model would not be able to know what a user's main task was without prior knowledge. Based on detailed inspection of video segments corresponding to Coarse breakpoints, we observed that they were frequently tied to switches among various types of applications or content, e.g., music players, e-mail and instant messaging, or online shopping and news. Our observations are also consistent with results derived from an analysis of users' activity data, as reported in (Czerwinski, Horvitz et al. 2004).

We thus created a set of application categories including Entertainment, Communications, and Web; with the latter being further categorized based on whether it is a common news or shopping site based on its URL; and those already being used as part of this work (DE, IM, and P). Under the assumption that various applications could be mapped to these categories, features were created for the number of switches between

General (Coarse)
#CompletedOpenAnyApp
#SwitchToEntertainmentApp
#SwitchToOnlineNews
#SwitchToDocEditing
#SwitchToImageManipulation
#SwtichToProgramming
#SwitchesToCommunications
#CompletedStartAnyApp
#CompletedMaximizeAnyApp
#CompletedExitAnyApp
#CompletedRelocation

Table 6.6: . A representative sample of the candidate features used for detecting Coarse breakpoints. This is across all tasks. The number of occurrences of each feature were counted for each 10s bin of task execution. The features highlighted in bold were found to be predictive.

them. Also, the number of applications started, exited, and moved were included, as these have also been argued to indicate switches in high-level activity (Nair, Voida et al. 2005).

Though our approach offers a reasonable starting point and extends prior work for detecting Coarse breakpoints, future work should explore the value of including features tied to the degree of similarity among application content, e.g., using techniques in (Baeza-Yates and Ribeiro-Neto 1999). Note that overcoming challenges of applying such techniques within the domain of interactive applications is well beyond the scope of our current work.

Medium and Fine breakpoints typically occurred during the interaction within an application. Our approach here was to bind features to independent actions at the application interface level, following work in (Fogarty, Ko et al. 2005). For example, for DE, features included CompletedSwitchToAnotherDoc, CompletedSetInsertionPoint, and CompletedScroll. If the first two occurred within a bin, then this would likely indicate Medium; whereas if the latter two occurred, then this might indicate Fine, e.g., due to switching paragraphs.

Document Editing 33 total	**Image Manipulation** 33 total	**Programming** 42 total
#CompletedFormattingActions **#CompletedSwitchToDocEditing** **#CompletedAlt-tabSwitch** **someKeystrokes** **noMouseClicks** **noMouseMoves** **someMouseMoves** #CompletedSwitchToAnotherDoc #CompletedSetInsertionPoint #CompletedSelections #CompletedSaves	**#CompletedAltTabSwitch** **#CompleteSwitchToOtherImg** **#CompletedSave** **#CompleteColorManipulation** **#CompletedTextManipulation** #CompletedSetupNewImage #CompletedExitCurrentImage #CompletedSelectionTools #CompletedLayerManipulation #CompletedCanvasResize #CompleteSelectionToolActions	**#CompletedOpenAnyApp** **#CompletedSearch** **#CompletedSwitchClass** **#CompletedSwitchProject** **#ControlKeyStrokes** **noMouseClicks** #CompletedSetInsertionPoint #CompletedSwitchMethod #CompletedCreateMethod #CompletedCreateClass #CompletedDebug #CompleteNavigateCode

Table 6.7: A representative sample of the candidate features used for detecting Medium and Fine breakpoints. This is across all tasks. The number of occurrences of each feature were counted for each 10s bin of task execution. The features highlighted in bold were found to be predictive.

For Coarse breakpoints, we identified 20 features that were independent of any one application. For Medium and Fine, we identified 33 features for DE, 33 for IM, and 42 for Programming, with some overlap. Samples of the features (with mnemonic descriptions) are provided in Table 6.6 and Table 6.7. One characteristic of many of the

features is they correspond to *completion* of an action, not the action itself (e.g. completed scrolling as opposed to scrolling), which is consistent with observers' explanations and the notion of a breakpoint.

A coding agenda was developed, comprising a description, example, and rule for each feature (Altheide 1996). For each breakpoint bin (10s clip), values for the features were computed by applying the agenda to corresponding parts of the videos. We also computed values for the features for a sample of bins that had *no* breakpoint (NAB), enough to compose 25% of the total training cases. Training cases were in the form of <value of feature 1, …, value of feature N, output>, where output was one of Coarse, Medium, Fine, or NAB.

The coding was validated by having an independent coder compute values for the candidate features for 10% of the bins, randomly selected from the training cases. Cohen's Kappa showed satisfactory agreement between them (0.74).

6.7 Extract Predictive Features

Before predictive features could be extracted, we needed to decide how the models would be built. Our approach was to create one application-independent model for predicting Coarse/NAB and a set of application-specific models for predicting Medium/Fine/NAB, giving a total of 4 models. This decision was made because Coarse breakpoints were deemed independent of any one application while Medium and Fine were more dependent. Training cases were organized accordingly, but Medium and Fine cases from each task category were included as part of NAB cases for Coarse, helping to minimize overlap between the models.

Given this organization of the training cases (models), the predictive features were extracted using Correlated Feature Selection (CFS) with a Greedy Stepwise search (Hall 2000). CFS was chosen since some candidate features may have been correlated. Predictive features are shown in bold in Table 6.6 and Table 6.7.

6.8 Map Predictive Features to Breakpoints

The last step was to learn models that map the predictive features to the breakpoint types and NAB. A multilayer perceptron (MLP) was leveraged to learn each mapping, as it does not assume independence of features and has been used to learn similar models in prior work (Iqbal and Bailey 2006). The model for Coarse breakpoints had two outputs

		Predicted		
		Coarse	NAB	Total
Actual	Coarse	62 (89.9%)	7 (10.1%)	69 (100%)
	NAB	11 (15.7%)	59 (84.2%)	70 (100%)

Table 6.8: Predicted vs. Actual for Coarse breakpoints. Overall accuracy was 87.1%.

(Coarse, NAB) while the models for each category of task had three outputs (Medium, Fine, NAB). All models had one hidden layer.

For input, the model for Coarse used only those features that were independent of the task (Table 6.6) while inputs for the other models corresponded to features tied to the application, in addition to the general features. Mappings were learned using back propagation, and a 10-fold cross validation was used to evaluate the models.

Table 6.8 shows results for predicting Coarse and NAB. The model yielded an overall accuracy of 87.1%, which is much better than the baseline ($\chi^2(1, 139)=76.3$, $p<0.001$; baseline =50%), where baselines were calculated as the accuracy of always predicting the most common outcome. The high accuracy can likely be attributed to the model's features detecting a switch between certain application categories that often indicated a switch between unrelated activities. More sophisticated analysis of the similarity between the content of applications may yield further improvements.

Table 6.9, Table 6.10 and Table 6.11 and show results for detecting and differentiating Medium, Fine, and NAB for the three task categories. For Document Editing, the model yielded an overall accuracy of 69.4%, which is much better than the baseline ($\chi^2(1, 85)=33.5$, $p<0.001$; baseline=39%). The model was slightly less accurate for

		Predicted			
		Medium	Fine	NAB	Total
Actual	Medium	20 (60.6%)	11 (33.3%)	2 (6.1%)	33 (100%)
	Fine	5(20.8%)	15 (62.5%)	4(16.6%)	24(100%)
	NAB	2(7.2%)	2(7.2%)	24 (85.7%)	28(100%)

Table 6.9: Predicted vs. Actual for Medium and Fine breakpoints during the Document Editing Task. Overall accuracy was 69.4%.

differentiating between Medium and Fine. However, the most egregious type of error, detecting either type of breakpoint when none existed was low (14.4%).

For Image Manipulation, the model yielded an accuracy of 76.3%, much better than the baseline ($\chi^2(1, 152)=42.1$, $p<0.001$; base=50%). This model was able to effectively differentiate Medium and Fine, and Medium and NAB. However, the model would sometimes predict Fine when the actual was NAB. This could be due to the mouse movements being less predictive of users' intents or there being less visible structure in this particular task category.

For Programming, the model yielded an accuracy of 75.8%, which was better than the baseline ($\chi^2(1, 91)=23.3$, $p<0.001$; base=51%). The model was slightly less effective at differentiating Fine and NAB, but it was very effective at differentiating Medium and NAB, and Medium and Fine.

Our models were developed using breakpoints identified by observers who did not share users' internal understanding of their tasks. As a final evaluation metric, we thus wanted to test how well our models could predict breakpoints identified by the users themselves. We asked users whose interaction data was originally annotated by observers to identify

		Predicted			
		Medium	Fine	NAB	Total
Actual	Medium	24 (68.6%)	10 (28.6%)	1 (2.8%)	35 (100%)
	Fine	5 (6.6%)	71 (93.4%)	0 (0%)	76 (100%)
	NAB	2 (4.9%)	18 (43.9%)	21 (51.2%)	41(100%)

Table 6.10: Predicted vs. actual breakpoints for Image manipulation. Overall accuracy was 76.3%

		Predicted			
		Medium	Fine	NAB	Total
Actual	Medium	11 (68.8%)	4 (25.0%)	1 (6.3%)	16 (100%)
	Fine	1 (2.2%)	36 (78.2%)	9 (19.6%)	46 (100%)
	NAB	0 (0%)	7 (24.1%)	22 (75.9%)	29 (100%)

Table 6.11: Predicted vs. actual breakpoints for Programming. Overall accuracy was 75.8%

breakpoints in their own data, and then tested the accuracy of our models on it. Applying our models to the user's annotated data sets, the accuracy of the model for Coarse breakpoints ranged from 40–100%, with an average of 76.5% across users (one user's data was excluded as too few breakpoints were identified). For the application specific models, the results for each user were (DE1: 56.0%, DE2: 72.7%; IM1: 68.2%, IM2: 85.7%; P1: 14%, P2: 50.0%). Other than for P1, these results show that our models were able to accurately predict breakpoints identified by the users, even though a number of these breakpoints did not intersect with those identified by the observers. This validates that our models can predict breakpoints independent of the knowledge of the task.

Overall, even though there were some errors, our results demonstrate that it is feasible to build models that detect and differentiate breakpoints within free-form tasks with fairly high accuracy. This ability to detect a majority of the breakpoints should be more than sufficient to allow useful functionality, e.g., to enable defer-to-breakpoint policies. Potential solutions for meaningfully improving the accuracy of the models involve identifying and integrating additional predictive features into the models, training the models for specific users, and experimenting with various bin sizes.

6.9 Discussion

This research sought to further understand different types of breakpoints across various tasks and examine the feasibility of building models that could detect and differentiate them.

Our work has produced several important findings. First, we were able to identify interactions that characterize each type of breakpoint. For example, a switch in high-level activity corresponds to a Coarse breakpoint, a switch in the current source object (e.g., document, image, or code file) of an application corresponds to Medium, and a switch in the action on the current object corresponds to Fine. This shows that there is a perceivable structure within free-form tasks, which models should be able to detect. Interestingly, these characteristics closely parallel those found to indicate breakpoints within physical tasks (Zacks and Tversky 2001).

Second, we found that temporal distances between types of breakpoints ranged from about 1 to 10 min, with an average of about 4 min. Our results support previous work

showing that users repeatedly multi-task (Gonzalez and Mark 2004), but also show that this multi-tasking occurs at multiple levels of detail. Our results also establish that breakpoints occur often enough such that interruption management systems could practically employ defer-to-breakpoint policies for non-critical notifications.

Third, we reported which features of user interaction were found to be predictive of each breakpoint type. Though our set of features should by no means be considered final, they do provide deeper insights into the range of features that should be included in similar models deployed in practice.

Finally, our models were able to accurately detect 69 to 87% of each type of breakpoint for the observers' data, and similar results were obtained for users' own annotations. We believe these results are very positive, especially since no pre-defined specifications of tasks were used. Increasing accuracy would likely require identifying and integrating additional predictive features, e.g., indicating similarity of content, or refining the default models for specific users.

6.9.1 Deploying Models for Detecting Breakpoints

To deploy similar models in practice, one must consider (at least) how to instrument applications, which breakpoints to detect and how to train the models, and what bin size to use.

Instrumentation of applications is needed to send relevant events to a model. Such instrumentation can be achieved by leveraging existing research efforts such as (Dragunov, Dietterich et al. 2005), intercepting application events by writing plug-ins (Bailey and Konstan 2006), or adapting the underlying UI toolkit (Fogarty, Ko et al. 2005). Regardless of the method used, our work provides valuable insights as to the type and level of detail of the instrumentation needed.

Our work shows that three types of breakpoints can be detected, but this does not mean that models must detect all three in practice. For example, in interruption management, if users are able and willing to have information deferred up to 4-5 min on average, then a system may only need to utilize the one model for detecting Coarse breakpoints.

Default models could be deployed and provide reasonable accuracy, but, if needed, accuracy could be improved by refining models on a per user basis (Fogarty, Hudson et al. 2004). This could be achieved by leveraging toolkits for generating models on-the-fly

(Fogarty and Hudson 2007), assuming users would be willing to provide the necessary input. Further improvements would require identifying and integrating additional predictive features.

Finally, to detect breakpoints, a model must typically assess interaction within a fixed time window. Our work suggests a window of about 10s, but an implementation may need to experiment with different values, considering the tradeoff between computational overhead and discriminatory power.

6.9.2 Limitations

There are several limitations to the work. First, our work investigated breakpoints within categories of tasks that all required the generation or manipulation of content. Future work should thus study models for detecting breakpoints in other tasks, e.g., those that stress information-seeking.

Second, we analyzed about one hour of task execution data from each of six users. Thus, our resulting models are only able to accurately detect breakpoints within the range of interaction that was captured in our original data. As our work has now shown that building statistical models for detecting breakpoints is feasible, more robust models could be built by applying the methodology in this work on a much larger sample of interaction data.

Finally, the cognitive duration of a breakpoint is not known, but would be important for certain applications of our work, e.g., for interruption management where cost may not be reduced if information delivery exceeds this duration after a breakpoint. Future work should try to empirically determine this duration, which may inform the bin size for the models.

6.9.3 Linking Cost of Interruption to Perceptual Breakpoints

In this work we assume that different granularities of breakpoints correspond to different levels of interruption cost. The basis for this assumption comes from our investigation into mental workload patterns described in Chapter 4. In that work, we showed that the level of a breakpoint affected interruption cost; breakpoints higher in the task hierarchy had lower interruption costs than breakpoints lower in the task hierarchy. Here we make the tacit assumption that since structural levels of breakpoints and perceptual

granularities of breakpoints are often synonymous, interruption costs also differ across different breakpoint granularities. However, further studies are needed to verify this.

One way may be through applying a similar approach to the method for predicting cost using task structure, as described in Chapter 5. In this case, however, the mapping would be from characteristics of breakpoints or interactions surrounding breakpoints (as opposed to characteristics of the task structure) to different cost classes. The procedure would entail interrupting users at a representative sample of breakpoints during free form tasks, and collecting interruption cost data, i.e. resumption lag. Mappings between characteristics of each breakpoint and cost would then be learned.

As we have now gained valuable insights into how to identify breakpoints using statistical models, the next step is to realize these methods in an interruption management framework. We discuss such a framework in the next chapter.

CHAPTER 7

OASIS: An Omniscient Automated System for Interruption Scheduling

In this chapter, we present OASIS, a fully automated system that applies methods previously developed to schedule notifications at breakpoints during interactive tasks. OASIS stands for Omniscient Automated System for Interruption Scheduling. OASIS defers notifications until breakpoints during user tasks, where breakpoints are selected within a given time window specified by the application sending the notification. The benefit of using OASIS is that disruption caused by the interruption from the notification is reduced, without compromising the timely delivery of information. Additionally, a fully operational system allows us to test various defer-to-breakpoint policies in practice, which until now has not been possible.

We first present an example scenario illustrating the high level operations of OASIS, followed by a discussion on the various defer-to-breakpoint policies that OASIS currently supports. We then provide a detailed description of the system architecture of OASIS and describe how OASIS supports development of statistical models of breakpoints. We conclude with a discussion on deployment of OASIS in real life settings.

7.1 Example Scenario

The goal of OASIS is to detect when breakpoints occur during interactive tasks, and inform applications that are trying to gain the user's attention. Applications specify *defer-to-breakpoint* policies which OASIS takes into account while informing applications about breakpoints. Policies consist of a preferred notification delivery moment, e.g. a certain type of breakpoint (coarse, medium or fine), any breakpoint or the coarsest breakpoint within a given timeframe etc.

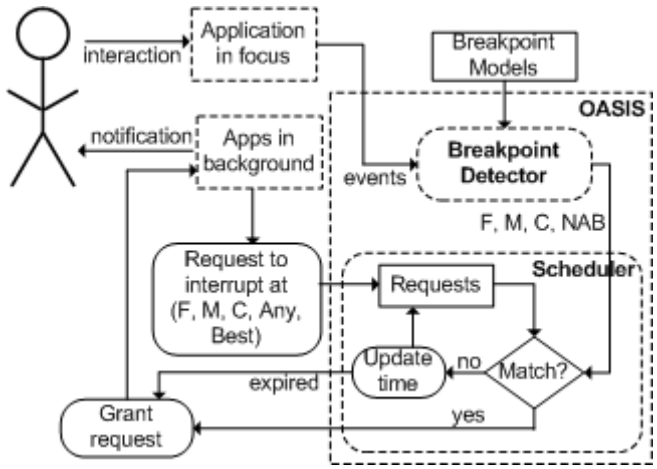


Figure 7.1: Schematic of Oasis. Applications send a request to Oasis, which adds the request to the queue. When a breakpoint occurs, the Breakpoint Detector informs the Scheduler. The Scheduler matches the breakpoint against the breakpoint specified in the policy.

To illustrate the operation of OASIS, we describe an example scenario. An email client generates a 'new email' notification and sends a request to OASIS. The request consists of the preferred policy for notification delivery and maximum time that the notification can be delayed. In this case, the policy selected by the email client is *next-coarse* and the notification is to be delivered within the next 60 minutes. When OASIS receives this request, it adds it to a queue. This request will be granted when the next coarse breakpoint occurs.

A background component in OASIS, the Breakpoint Detector, monitors events and classifies moments into *Coarse*, *Medium*, *Fine* and *NAB* (not a breakpoint), using predefined models of breakpoints. The models are developed a priori, using a separate model development component in OASIS. When the Breakpoint Detector sends back a signal indicating that a breakpoint has occurred, OASIS checks its type against the policies of the pending requests in the queue. If there is a match, OASIS grants the request immediately. For now, we assume all pending requests matching the policy will be granted when there is a match. However, we investigate this design choice in the evaluation of OASIS, presented in the next chapter.

For other requests still pending, OASIS waits for the next breakpoint. Coming back to the original example of the email notification request, if a coarse breakpoint did not occur within the specified timeframe, then the request is granted at the end of the 60 minutes.

This is under the assumption that notifying at any point within the requested timeframe would have the same benefit for the user. Figure 7.1 illustrates the process.

7.2 Scheduling Policies

A scheduling policy specifies the type of breakpoint the current notification needs to be delivered at, often subject to additional criteria being fulfilled. The scheduling policy is specified in the request an application sends to OASIS. Here we discuss the scheduling policies OASIS currently supports. All policies have an associated timeframe within which the notification must be delivered to retain its maximum benefit. If there is no breakpoint matching the stated policy within the timeframe then the notification is delivered as soon as the timeframe expires.

- **Interrupt at the next Coarse**

This policy requires that notifications be deferred until a Coarse breakpoint, so the scheduler withholds granting the request until the Breakpoint Detector sends a Coarse breakpoint signal. We envision this policy to be used for notifications that are less relevant or urgent and are typically of general interest that the user can attend to at leisure.

- **Interrupt at the next Fine**

This policy requires that notifications be deferred until a Fine breakpoint. This policy is useful for more urgent notifications that should not be delayed until a less frequent Coarse or Medium breakpoint. Since fine breakpoints occur more frequently than both Medium and Coarse, this policy ensures that notifications are delivered to users with little delay. This policy would be particularly useful for urgent notifications.

- **Interrupt at the next Medium**

This policy requires that notifications be deferred until a Medium breakpoint. This policy may be useful for notifications that are relevant to the ongoing task so that they should not be delayed until a Coarse, but not so urgent that they need to be delivered at a Fine.

- **Interrupt at the next breakpoint of any type**

This policy requires that notifications be deferred until the next breakpoint, be it Coarse, Medium or Fine. This policy ensures that a notification is delivered at a perceptual break, at any level of granularity.

- **Interrupt at the best breakpoint within a given timeframe**

This policy requires that notifications be delivered at the best breakpoint, within a given timeframe. In most cases, the coarsest breakpoint will be the best breakpoint. This is the most complex policy, as it requires the system to forecast at every breakpoint, whether there will be a better breakpoint within the remaining time. If there is a high probability of a coarser breakpoint occurring in the time left, then the system waits. If the current breakpoint is the best in foreseen future, then the request will be granted immediately. This ensures that the notification is delivered on time, at a breakpoint that is predicted to be coarsest within the available time.

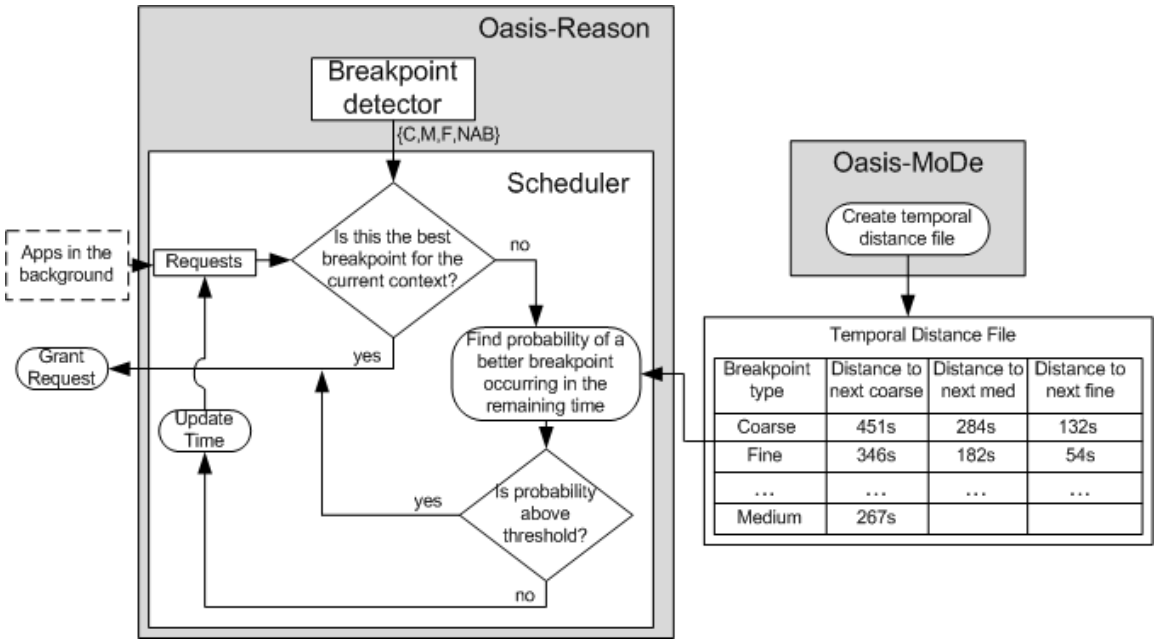


Figure 7.2: Illustration of how defer-to-best-breakpoint policy operates. The temporal distance file is created during the model building process and made available to Oasis-Reason. The scheduler checks the context (related to application, general interest) of the most current request and matches it against the current breakpoint. If the breakpoint is the best breakpoint within the context, it grants the request. If not, it uses the temporal distance file to determine the probability of a better breakpoint occurring within the remaining time for that request. If the probability is above a threshold, the scheduler waits, otherwise it grants the request.

Figure 7.2 demonstrates this process. During model building, temporal distances between a breakpoint and the next occurrence of each type of breakpoint (e.g. coarse to next coarse, coarse to next medium, coarse to next fine etc) are saved in a separate file. In real time, in order to decide whether to grant a request at the current moment or to wait for a better breakpoint, probability distributions are created. For example, if the current breakpoint is a Medium, and there are 50 more seconds left for delivery of the notification, then the probability that a better breakpoint occurs within the 50 seconds would be calculated as:

P(coarse occurring within the next 50 seconds|current BP=med)

= # (med-> next coarse pairs|dist <=50)/total # (med->coarse pairs)

This means that given the current breakpoint is a medium, the probability of a Coarse breakpoint within the next 50 seconds, is the number of *medium to next coarse* distances (in the temporal distance file) that are less than or equal to 50s divided by the total number of *medium to next coarse* pairs. More formally, this can be expressed as:

P(coarser bp occurring within X sec | current bp=med/fine)

= #(current bp -> coarser bp pairs|dist<=X)/total#(current bp -> coarser bp pairs)

If the probability is greater than some predefined threshold, then the system will wait for a coarser breakpoint to occur. Otherwise it will grant the request right away. The threshold can be determined based on whether the system wants to be more conservative in its estimation, resulting in higher thresholds; or less conservative, resulting in lower threshold. Determination of the ideal threshold is an interesting problem and opens up avenues for future research.

OASIS currently supports two domains – programming and diagram editing, but can be extended to support others. This allows us to first demonstrate the effectiveness of the system, without expanding the effort to instrument many applications. Microsoft's Visual Studio and Visio were chosen as the specific applications for programming and diagram editing, respectively. The broader vision, of course, is to eventually instrument most common applications in the desktop to report events to OASIS and provide a more comprehensive assessment of when breakpoints occur in practice.

7.3 Reasoning Component of OASIS

OASIS -Reason is the real-time reasoning component of OASIS, responsible for detecting and differentiating breakpoints, and scheduling notifications to occur at breakpoints based on the aforementioned breakpoint policies. It consists of two sub-components –the *Scheduler*, which interfaces with external applications and informs them when an appropriate breakpoint occurs, and the *Breakpoint Detector*, which sends a continuous stream of breakpoint events to the Scheduler. Figure 7.1 provides a schematic of the operations of OASIS -Reason.

7.3.1 Scheduler

The scheduler accepts notification requests from external applications and schedules them to occur at breakpoints according to policies specified in the notification request. Each request consists of a policy and a time window, by which the notification has to be delivered in order to maintain full benefit to the user. Determination of the length of the time window requires consideration of urgency and relevance of the notification in context of the user's ongoing task and is beyond the scope of this work. We assume that applications will have algorithms that code the relevance and urgency into a time window before sending notification requests to the scheduler.

On receiving a request, the scheduler first adds the request to an internal queue, sorted based on time left by which the notification must be delivered. It then establishes a connection with the Breakpoint Detector, if one has not already been set up. The Breakpoint detector sends breakpoint information to the scheduler as they occur during the user's task execution. Upon receiving a breakpoint notification, the Scheduler checks whether the breakpoint matches the criteria specified in the policy for the first request in the queue. If there is a match, then the request is granted immediately and the corresponding application renders the notification for the user. If there is no match, the Scheduler updates the time left for the notification requests in the queue and listens for the next breakpoint signal to arrive. If time expires for any notification, the request is granted at once, ensuring that the notification is generated within the given time limit and the benefits of the notification is preserved.

7.3.2 Breakpoint Detector

The goal of the Breakpoint Detector is to continuously monitor a stream of user activities and using supplied models, determine whether the current moment is a breakpoint, and its type. Models are pre-created using the Model Builder component of OASIS.

The breakpoint detector detects two types of events: application events, which are known to be predictive of Medium and Fine breakpoints, and system event data, which are known to be predictive of Coarse breakpoints. Application event data is monitored through custom plug-ins. We have created plug-ins for two applications: Microsoft Visual Studio 2005 and Microsoft Visio 2007. Similar plug-ins can be generated for a much wider variety of applications providing a richer source of evidential data for breakpoint identification.

Data is collected based on an event-driven approach, where occurrences of registered events raise corresponding event handlers that log the event, time of the event, corresponding interface status on occurrence of the event (e.g. name of the application in case of an Open_Window event). Events that occur in large batches in succession (e.g. consecutive mouse events) are aggregated before saving. For example, moving the mouse cursor from one location to another will be saved as a single mouse move event, logging as well the start position and the end position, rather than logging each intermediate mouse movement, which has little informative value. Among the window status events, window creation, destroy, move, resize are logged along with the name of the underlying application and partial name of the window itself. This information is collected since it provides important insight into when application level switches occur, which has been found to be predictive of coarser level breakpoints in our prior work. In order to prevent collection of sensitive textual information, the keyboard input is masked before saving except for hot-key sequences such as copy, paste, save, open etc and arrow keys.

Although the plug-ins are provided with the capability of logging a wide variety of events, only those events that are found predictive within the predictive models are monitored and logged in practice. This minimized the amount of system resources engaged for monitoring breakpoints The logged events are compiled into test cases, similar to the training examples used to create the models. Test cases are created at 3

second intervals, and evaluated using the models. This provides a continuous stream of classification data, belonging to either of {Coarse, Medium, Fine, NAB}.

The process of evaluating a moment is illustrated in Figure 7.3. As a user performs tasks in an instrumented application, a custom plug-in sends callbacks corresponding to predictive events to an event handler in the Breakpoint Detector. Only those events that are predictive within the breakpoint model are monitored and logged by the plug-ins. The logged event is then added to a data structure. Every third second, a test case is created from the data structure and evaluated using the models. Evaluation is set up so that the test case is first evaluated using the global model, to determine whether the current moment is a Coarse breakpoint. If not, the test case is then evaluated against the application-specific model to determine whether it is Medium or Fine. If it is neither, then the current moment is classified as a NAB.

OASIS-Reason provides separate sockets listening for events in each instrumented application within the user's task environment as well as the global windowing events,

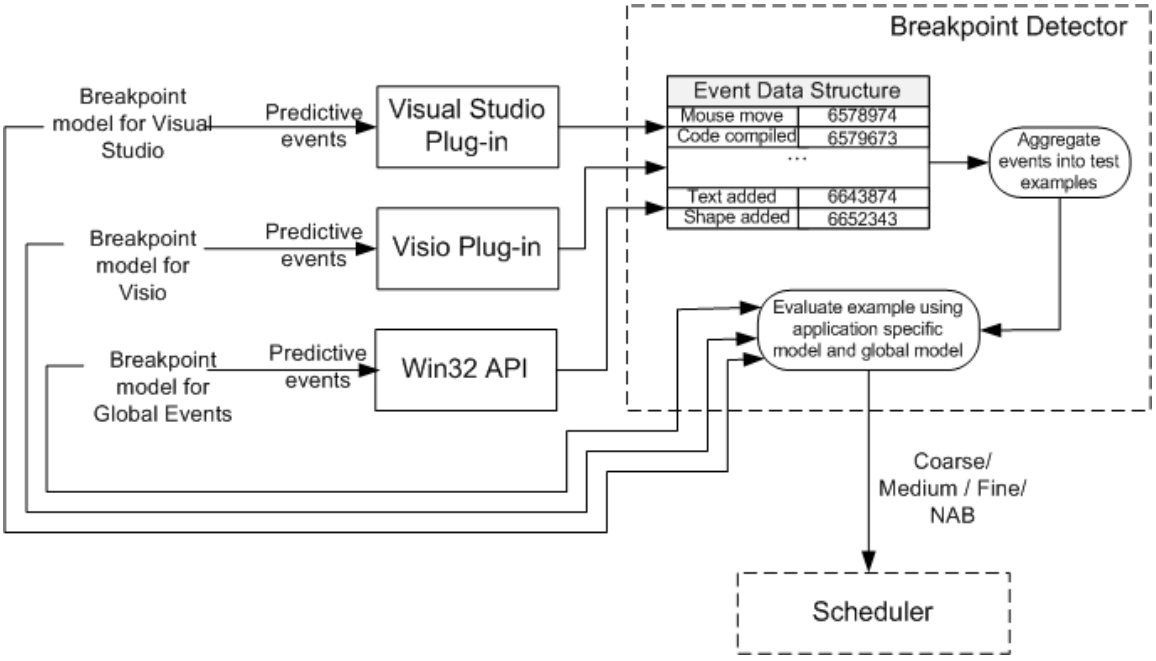


Figure 7.3: Schematic of the Breakpoint Detector. Visual Studio and Visio are shown as example applications being monitored. Events are aggregated into examples which are then evaluated using the global model (for Coarse) and the relevant application specific model. The result of the evaluation (Coarse/Medium/Fine/NAB) is passed on to the Scheduler.

and sends events to an event data-structure. The data structure is flushed every 10 minutes, in order to prevent overflow. The choice of three second intervals for evaluation was mainly to account for delays in aggregating data from multiple sources and evaluating events using the models on a standard 1.87 MHz Pentium 4 PC. Pilot studies showed that the three second interval is sufficient, since breakpoints typically span more than three seconds and it is less likely that a true breakpoint will pass by undetected. However, with faster processors this interval can be reduced to less than a second, though there was no evidence suggesting the need to do so at this conjecture.

The output of the breakpoint detector is a continuous stream of classification data in three second intervals. The data is used by the scheduler to determine when to send a clearance signal to external applications intending to send notifications to users.

7.3.3 Instrumentation

For Visual Studio® 2005, we created add-ins that extended the functionality of the IDE so that most events within the IDE would be logged. This was done by implementing the IDTExtensibility2 interface and writing code for a number of event-handlers that saves the event name, time of events and peripheral information such as project name, file name etc. For Visio® 2005, we used Microsoft® Visual Studio® 2005 Tools for the 2007 Microsoft Office System (VSTO 2005 SE) which provides support for application-level add-ins for the applications in the 2007 Microsoft Office System. VSTO 2005 SE is based on Microsoft .NET Framework 2.0 and Visual Studio 2005. Our add-in for Visio 2007 is written in managed code, exposing public methods OnStartup and OnShutDown and public events Startup and Shutdown. The Visio application exposes an Application Object, which is accessed by custom code in the derived class of the add-in code to achieve desired functionality. Our add-in code logs events corresponding to most actions in the application, starting from opening new/existing diagrams to making modifications in shapes/diagram/text, adding new pages, saving documents etc. To log the data, an event based approach is followed as before, where event-handlers are set up that save information on the event, time of event and relevant information such as item name or size. To monitor system events, the User Interface functionality of the Win32 API has been leveraged to collect information on screen windows, and mouse and keyboard input.

Applications wanting to send notifications connect to the Scheduler through an IP Version 4 stream socket using the TCP/IP protocol. Typically notifying application clients will be on the same machine as OASIS, therefore the connection is established using the loopback address 127.0.0.1 as default. However, this address can be replaced by any other IP address if OASIS and the notifying application reside on different machines. Applications that can potentially send notification requests set up connections with the Scheduler a priori. The connection request is sent to a predetermined port, through a non-blocking *send*, using an asynchronous callback. This allows applications to be notified to take appropriate action if the connection fails, without blocking on the request.

If the connection is successful, the application uses this connection for sending future notification requests. A notification request consists of a policy and a timeframe (null if the policy is not the 'best breakpoint' policy) and is converted to a bytearray before transmission. On a successful transmission, the request is queued at the Scheduler, which waits for the Breakpoint Detector to send the appropriate breakpoint notification.

7.4 Model Development Component of OASIS

OASIS-Mode is the offline model development component of OASIS, comprising of the following sub-components: Activity Recorder, Breakpoint Annotator, Data Post-processor and Model Builder. Models can be developed on a per-user basis, where each individual records their activity data and provides their own annotations to build the breakpoint models. The other alternative is creating generic models, where breakpoint

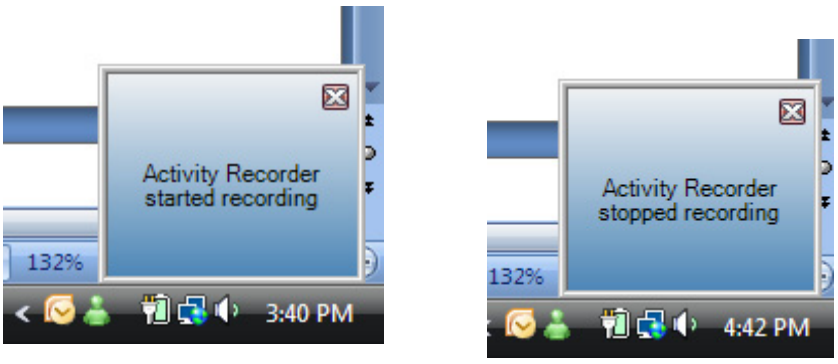


Figure 7.4: The Activity Recorder executes completely at the background of user attention through worker threads. The user is informed only when the recording starts and the recording stops through pop-up windows near the system tray.

annotations data from a variety of users are combined to provide training examples for models. Each type has its tradeoffs and we will discuss these in the discussion section. The basic mechanism, however, is same for both types, which we discuss in the following sections.

7.4.1 Activity Recorder

The goal of the Activity Recorder is to record interactive task execution data, which will later provide contextual samples of breakpoints and associated events. The Activity Recorder collects three types of data: onscreen video streams of visible user activities, application level events corresponding to application specific activities and global events corresponding to window manipulation in the GUI. Video streams are collected so that they can be used in retrospective coding of breakpoints, providing a playback of the user actions and allowing identification of breakpoints. Application and window events are collected to generate features that may be predictive of breakpoints. Since a user's actions often represent their cognitive goals, we believe that if the proper events are logged, features can be generated that can predict breakpoints with reasonable accuracy.

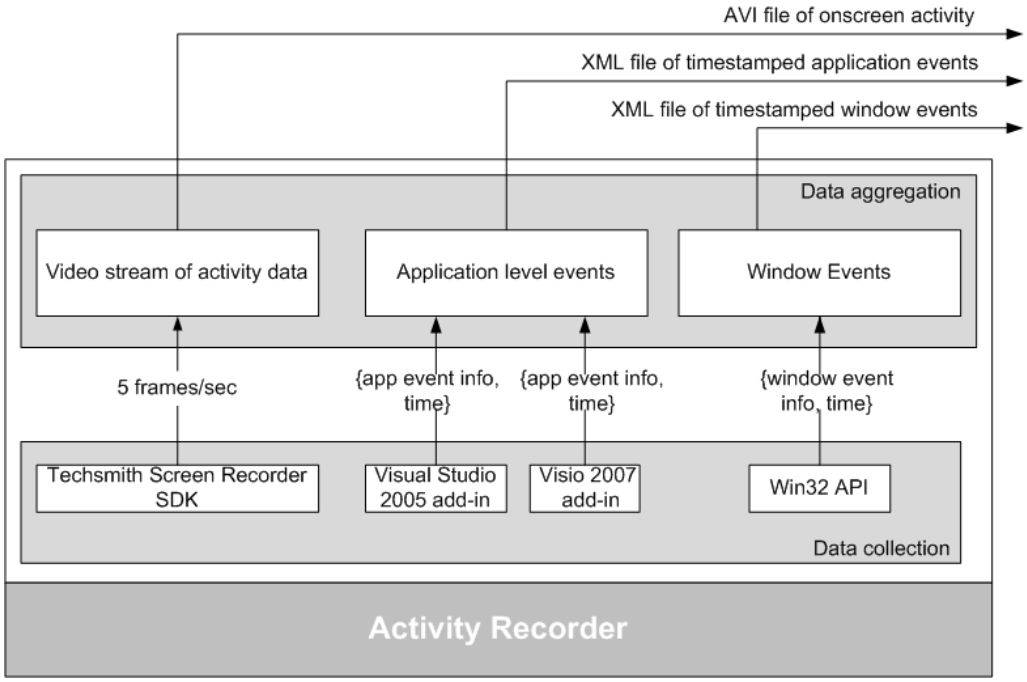


Figure 7.5: System architecture of the Activity Recorder. Data is collected through lower level APIs and add-ins. Collected data is aggregated as AVI and XML files when recording is stopped. The collected data files are used by the Breakpoint Annotator and the Data processor in later phases.

Figure 7.5 shows the system architecture of the activity recorder. Video data is collected using the Techsmith Screen Recorder SDK, by making standard DLL calls to the library's single 32 bit DLL. Video frames are captured at a rate of 5 frames/second. This frame rate ensures that the video quality is high enough to capture ongoing activities without consuming too much system resources. The recording of the video operates in the background through worker threads asynchronously from the calling code, ensuring that other activities can be carried on without waiting for the current frame to be captured. Videos are saved as AVI files in order to achieve maximum portability.

Event data is collected from custom plug-ins for Visual Studio and Visio as well as global windowing events, similar to those used in the Breakpoint Detector. At the end of the data collection phase, the video recordings are saved in AVI format for portability and the data logs are saved as separate XML files for global and application level events. The AVI file is used by the Breakpoint Annotator, while the Data Post-processor uses the XML files to associate events with breakpoints.

7.4.2 Breakpoint Annotator

The Breakpoint Annotator allows a user to view a video of user activities and annotate where they feel breakpoints exist in the viewed activity sequence. The idea of the Breakpoint Annotator has its roots in the 'Unit Identification' process of Heider (Heider 1958), later adopted by Newtson et. al. (Newtson 1973; Newtson and Engquist 1976; Newtson, Enquist et al. 1977) and Zacks and Tversky (Zacks and Tversky 2001) to segment streams of user actions into stand-alone behavioral units. Allowing retrospective coding to identify breakpoints has two advantages. One, it allows people other than the users themselves to apply their domain knowledge to identify breakpoints in the observed sequence of events. This is important, since these independent observers do not possess the internal knowledge that the users have to better understand where breakpoints exists – and this is representative of the situation that an automated system will have to model. Therefore, retrospective coding of the videos allows the observers to leverage only the observable events to help identify breakpoints, which would be similar to the evidence available to an automated system. Two, if the users themselves do identify their breakpoints, we believe that the retrospective coding is still a more appropriate

mechanism for identifying breakpoints, since it does not interrupt the user's normal course of activities as would techniques such as experience sampling would have done.

Image or video based reminders has been found to be an effective mechanism in recreation of past experience (Sellen, Fogg et al. 2007). The downside, however, is the fact that the context may not be readily or accurately recalled during retrospective coding, which could potentially result in erroneous identification of some breakpoints. However, since our eventual goal is to reduce the breakpoint coding burden by using generic models expediently across many users, the issue of losing context becomes somewhat

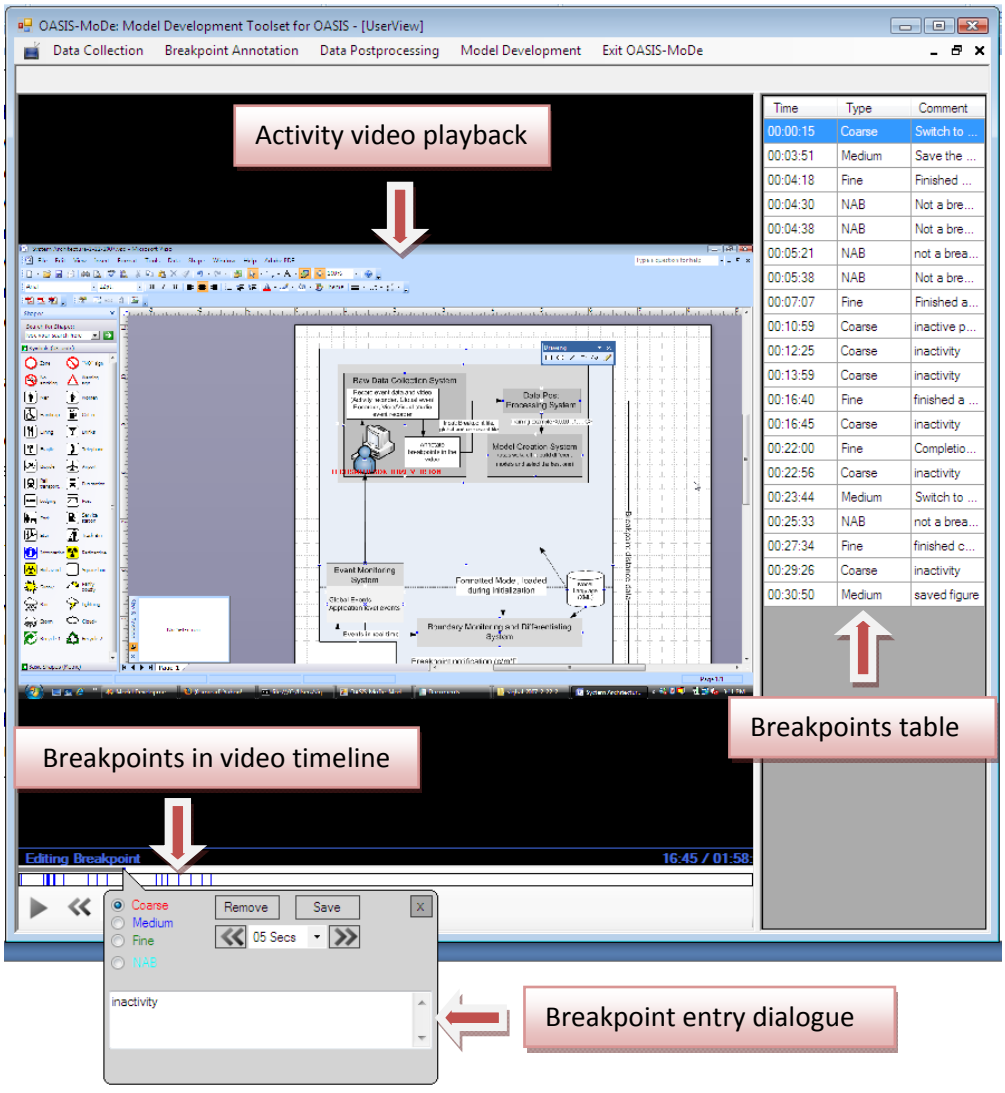


Figure 7.6: The Breakpoint Annotator provides a combined interface for annotating breakpoints and visualizing breakpoint position in terms of the video timeline as well as the content of each breakpoint.

moot, since the person identifying breakpoints retrospectively would likely not be the person who performed the task.

Breakpoint annotation can be done by either users or independent observers and noting which group the annotator belongs to is important for reasons discussed in the next section. The Breakpoint Annotator asks for this information when it is launched. To identify breakpoints, the annotator first imports the video into the Breakpoint annotator and starts playing the video (Figure 7.6). Controls on the video playback interface allow the annotator to stop, pause and play the video at any point. Frames can be rewound or fast-forwarded using the appropriate controls. To insert a breakpoint the user clicks on the 'Insert Breakpoint' button, which pauses the video and shows a modal dialogue box. The dialogue box provides four radio control buttons corresponding to *Coarse, Medium, Fine* and *NAB* and a text box for explaining why the annotator chose the selected breakpoint type for the current moment. Clicking on the save button adds the breakpoint information to a table on the right side of the video and starts playing the video onwards.

On completion of annotation, the annotated video file is saved as a persistent *.bpf file, which is a zip file consisting of the video and a time-stamped XML file consisting of the breakpoints. The bpf file can be loaded and modified at any point, allowing annotators to annotate breakpoints over multiple sessions. The bpf file can also be used by many annotators and an XML file is created for each person. During loading the file, any user can be selected and their saved breakpoints show up in the table for breakpoints in the Breakpoint Annotator interface.

The Breakpoint Annotator is implemented in managed code, written in the Visual C# language in the .NET Framework 2.0. Video playback controls are based ActiveX COM controls for Windows Media Player. The system uses wrapper DLLs (AxWMPLib and WMPLib) that automatically marshal between the .NET code and the ActiveX COM, allowing incorporation of Windows Media Player-like controls on to the Breakpoint annotator interface. The breakpoint information is saved as XML entries, leveraging the .NET 2.0 Framework XML classes. The video file and the breakpoint file are zipped together as a *.bpf file using the .NET 2.0 Zip library.

The *.bpf file is then passed on to the Data Post Processor, where breakpoint annotations are associated with events files from the Activity recorder to create training examples for the model building process.

7.4.3 Data Postprocessor

The Data Postprocessor creates training examples corresponding to breakpoint annotations created using the Breakpoint annotator, in the format of <f_1, f_2, f_3.... f_n, *breakpointType*>, where f_n denotes feature. These features are generated from the application and window events collected by the Activity Recorder.

The Data Postprocessor accepts three data files as input: the *.bpf file containing the breakpoints, the xml file containing the application level event information and the xml file containing the window event information. It determines what events have occurred prior to each breakpoint, associating breakpoints and events through common timestamps and distills the event information into discrete feature values.

The Data Postprocessor first creates a list of features, based on all possible raw events recorded by the Activity Recorder and also attributes derived from the raw events. Examples of raw events are *minimizeWindow, addNewItem, afterCompile, mouseMoved, mouseKeyDown* etc. Examples of derived events are: *lowMouseMovementRate, mediumMouseDirectionChangeRate, highNavigationRate, unrelatedSwitches, highInactivityRate* etc. A full list of features for Visual Studio and Visio as well as the global Window status are provided in Appendix A.

The Data Postprocessor counts the occurrences of all events (raw and derived) for different time windows preceding the breakpoint, where the time windows are: less than 1 second, 1-2 seconds, 2-5 seconds and 5-10 seconds prior to the breakpoint. This provides a better sense of the history of events before a breakpoint rather than the instantaneous occurrence of events just prior to the breakpoint, The counts are then appended together to create a list of features corresponding to the format of a training case with the breakpoint type as the output.

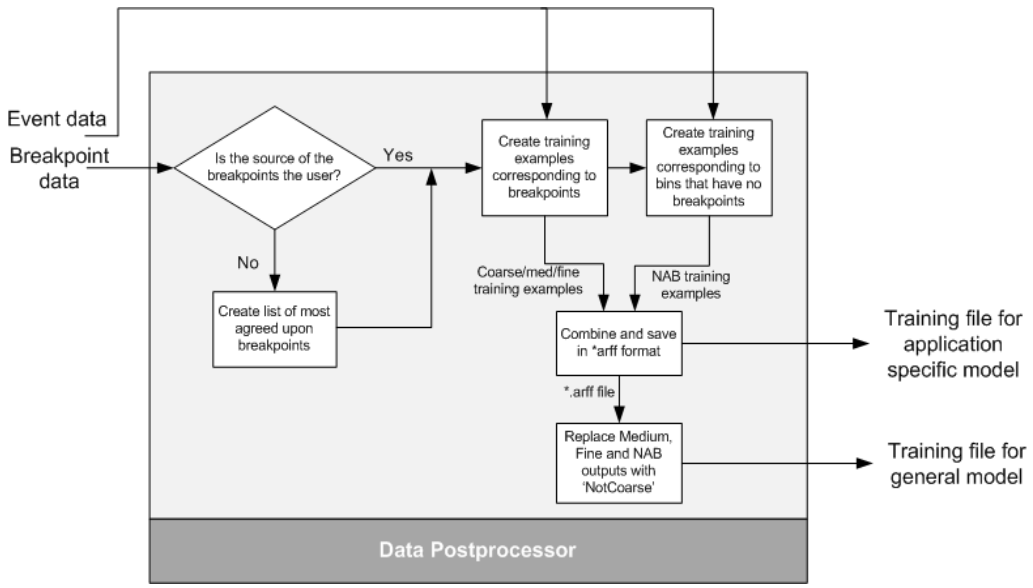


Figure 7.7: Underlying operations of the Data Postprocessor. Breakpoint data is associated with features derived from the event data and output in terms of two training case files, one each for the general model and for the application specific model.

In order to tell apart Non-breakpoint moments (NAB) from breakpoints, there needs to be additional training examples corresponding to NABs. Since NABs occur with a much higher frequency than do breakpoints, the Data Postprocessor automatically creates examples for moments that are not specifically marked as breakpoints and classifies them as NABs. Naturally, the number of NAB examples greatly outweighs the number of breakpoint examples, meaning that the models built will be more biased towards predicting a moment to be a NAB. However, our goal is to minimize the most egregious error of predicting NABs as breakpoints (potentially scheduling notifications to occur at those moments) and this approach supports that goal, although it may miss some breakpoints.

To create NABs, rather than looking at every second, the video sample is divided into 10-second bins and only those bins where there are no breakpoint annotations are used to create NABs. Since user activities do not typically change drastically in a 10-second granularity and since the generated features capture activities within a window 10 seconds prior to the breakpoint, we believe that this binning approach is adequate for capturing a representative sample of NAB training cases.

The Data Postprocessor also provides options for creating training examples from annotations from the users who contributed the activity video and corresponding events, or from a group of independent observers. Creating training examples from users is straightforward, since the user applies their internal knowledge of performing the task while identifying breakpoints and examples can be generated using breakpoints directly from the bpf file. For observers, however, creation of training examples is slightly more complicated. Since the observers do not have the advantage of experiencing the task themselves, they apply their domain knowledge and leverage visual cues from the activity video to guess where the breakpoints may have been. In order to reduce noise in this identification process, which can be highly variable due to different interpretations from different observers, the Data Postprocessor only selects breakpoints that have high degrees of agreement across observers.

Determining agreement across observers also requires careful deliberation since inherent differences in reaction time would likely cause observers to identify the same breakpoint at slightly different moments, rendering them to be different breakpoints. The Data Postprocessor circumvents this issue by using the same 10-second bin approach as before where all breakpoints in a 10 second bin are considered to be the same. The choice of 10 seconds is based on our prior research and appears to be sufficiently large so that same breakpoints are not spread across different bins, rendering them to be different breakpoints. At the same time, 10 second bins are small enough to ensure that different breakpoints are not lumped into the same bin. While creating training examples, the features are generated based on the breakpoint that is furthest down the line in the 10 second bin (see Figure 7.7), which takes into consideration the most events that could have potentially contributed towards the current bin being considered a breakpoint.

Since model performance depends on the quantity and quality of training examples, the Data Postprocessor also provides an option to combine multiple training case files originating from multiple activity data. This allows more variety and diversity in the examples of breakpoints, making the models more robust. For example, to create models for Visual Studio users, we merged training examples created from activity data from 3 different users, providing a model with far more coverage than a single model.

Based on our prior findings of Coarse breakpoints being application independent and Medium and Fine being application dependent, we designed the Data Postprocessor to create separate training case files for generating models for Coarse breakpoints, and for Medium and Fine breakpoints. The only difference in the training files is that training cases corresponding to Medium and Fine are annotated as '*Not Coarse'* in the training case file for the application independent (general) model. The reason for this is as follows: a breakpoint that is not a Coarse can potentially be a Medium, Fine or a NAB (Not a breakpoint). Any example classified as a Not Coarse will be therefore presented to the application specific model (for Medium/Fine) for further categorization.

Event data read from the files are stored in DataSet structures, which are in-memory databases in C# and can be accessed through regular SQL commands. Training cases are created using separate worker threads so that the system is still responsive while the training cases are being generated.

Training case files are formatted using the Attribute Relation File Format (*.arff) as required by the machine learning library from Weka. These files are then fed into the Model Builder.

7.4.4. Model Builder

The Model Builder accepts training example files from the Data Postprocessor and outputs models for predicting breakpoints. It runs the examples in the training case files through a number of machine learning algorithms in order to learn the best model that fits the data. The model builder also outputs a list of features that are predictive. Both the models and the predictive features are used in the real time component of OASIS.

The model builder accepts two training files and produces two models – a general model for differentiating between Coarse and Non-coarse moments, and an application specific model for further differentiating Non-coarse moments into Medium, Fine and NAB moments. The model builder begins with a feature filtering process, in order to determine the features that are most predictive and would contribute to a parsimonious model with maximum accuracy. A smaller number of predictive features would also ensure that the real time component would only have to monitor a smaller set rather than the many features originally monitored for building models, saving valuable system

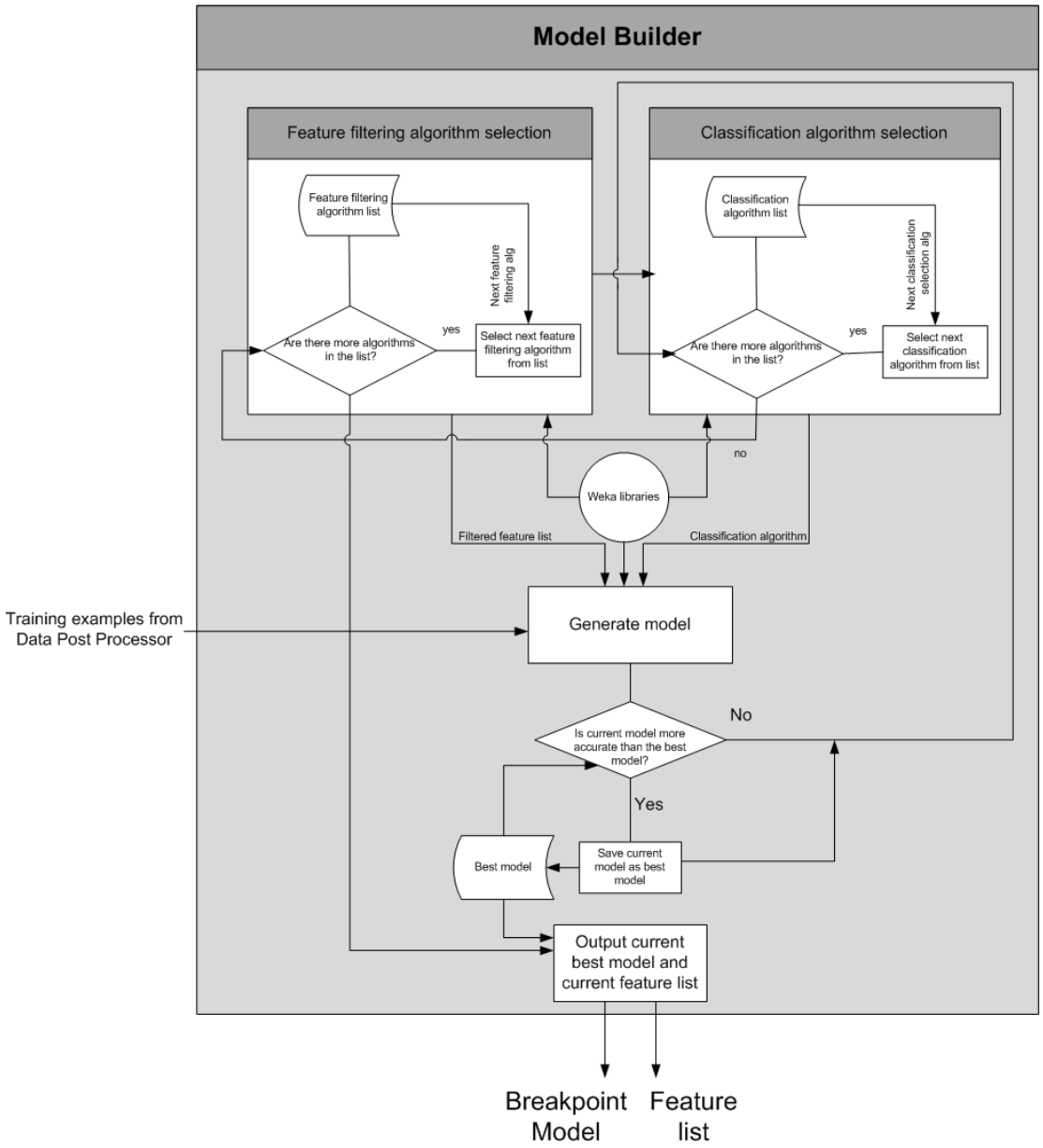


Figure 7.8: Underlying operations of the Model Builder. Training examples generated in the Data Postprocessor component are passed on to the Model Builder system. The training examples first go to the Feature filtering component, which selects and applies a filtering algorithm from a list of filtering techniques and then passes the filtered examples to the Classification algorithm component. For each filtering technique, the training examples are run through a number of classification algorithm and the best model in terms of performance is saved. Once all classification algorithms are exhausted the next filtering algorithm is applied and the process goes on until all filtering techniques are testing with all classification algorithms. The final output is a predictive breakpoint model and a list of predictive features.

resources. After selecting the predictive features, the training examples are run through a variety of learning algorithms including decision trees, multilayer perceptrons and IB1.

This allows exploration of a variety of models rather than sticking to a single learning algorithm and gives us more confidence in the models learned. See Figure 7.8.

The model builder leverages Weka, a collection of machine learning algorithms for data mining tasks, in particular using the Weka tools for classification. The Weka libraries are written in Java, which is ported as a DLL library through IKVM.NET. IKVM.NET provides a Java Virtual machine implemented in .NET, a .NET implementation of the Java Class libraries and tool that enabled Java and .NET interoperability. The static compiler ikvmc.exe is used to compile the weka classes and the jars into a .NET assembly. Models are saved in the weka model format to be later used in the real time reasoning component. List of features are saved as XML files.

7.5 Discussion

To date the major focus in interruption research has been on developing models to predict interruptibility. There are only a few systems that actually use these models to schedule notifications. OASIS implements techniques shown to work in previous studies. It provides a framework that can be used to both develop models and provide real time scheduling of notifications. The framework also allows models developed using other toolkits, such as Subtle (Fogarty and Hudson 2007) or Notification Platform (Horvitz, Koch et al. 2002), providing a test bed for testing a variety of interruption management policies in an operational system.

An important question is to decide whether to develop personal models for each user, or to develop composite models using data from a representative sample of users and use the same models for different users. While personal models can embody more individual patterns, the effort to develop these models is not trivial and it is not clear whether this effort will outweigh the benefits of the system. One alternative is to create composite models using data from many representative users. This captures a wide multitude of interaction patterns within the models, and reduces the model building effort since these models have to be created once. Although the accuracy of the models may be lower than personal models, empirical evidence suggests that with careful sampling of users, composite models can provide reasonable performance, sufficient to demonstrate effectiveness of the system using the models. Our vision is to use composite models as

default models included with commercial implementations of the system, with capabilities within the system to learn individual behavior and refine the models accordingly. With default models with reasonable performance, this will ensure that users can realize benefits of the system without the upfront model development effort. Online learning of individual behavior will continue to improve model performance. Determining how to develop models for intelligent systems has been a challenge within the community and developing default models for initial use has been a growing trend (Horvitz and Apacible 2003; Mutlu, Krause et al. 2007; Fogarty and Hudson 2007).

OASIS supports a small set of defer-to-breakpoint policies, but many more can be tested. Due to the dearth of working notification scheduling systems in practice, there is little understanding of how to determine the appropriate policy to schedule notifications. With OASIS, it is now possible to deploy various policies in real life settings to determine how policies can help strike a better balance between the cost and benefits of notifications.

The current implementation of OASIS only supports two applications, Visual Studio and Visio, but the broader vision is to instrument common applications in the desktop to report events to OASIS Reason and provide a more comprehensive assessment of when breakpoints occur. However, as a starting point, we chose to limit the instrumentation to two representative applications, since we would first like to demonstrate the efficacy of the system before putting in the effort to instrument and develop models for a wider variety of applications.

In the next chapter, we evaluate the impact of using OASIS.

CHAPTER 8

Effects of Intelligent Notification Management on Users and Their Tasks

In Chapter 7 we described Oasis, an automated system for scheduling notifications. In this chapter we focus on evaluating effectiveness of using this system. Our goal is twofold. First, we would like to evaluate how well the system performs in detecting and differentiating breakpoints during novel task sequences, leveraging breakpoint models developed a priori from interaction from different users performing different tasks. Novel task sequences are those that are generated by different users performing tasks from the same domain, e.g., multiple programmers doing programming tasks. This is important, since the overall effectiveness of our system depends on the ability of the models to detect breakpoints in practice. The feasibility of building statistical models that detect breakpoints in *authentic,* interactive tasks has been demonstrated in prior work, described in Chapter 6, but has not been empirically tested.

Second, provided that the models perform reasonably well, we would like to investigate the impact of automatically scheduling notifications at breakpoints in terms of task productivity and user affect. We also want to evaluate how users react to automated notification scheduling. This is crucial because the impact of *automatically* scheduling notifications has not yet been studied in context of authentic tasks. It is thus not known whether deploying such systems would indeed have utility for the user.

In this chapter, we report results from two user studies that directly address both of these challenges. Two complex task domains - diagram editing and programming – were used in the studies. Our first study tested the performance of composite models for detecting and differentiating three granularities (types) of salient breakpoints within novel task sequences. Our second user study investigated how scheduling notifications at different types of breakpoints affects users and their tasks relative to delivering them

immediately. Breakpoints were being detected in real time using the models from the first study. Users identified the type of the breakpoints retrospectively, as a means for effectively compensating for the outcomes of our first study.

8.1 Evaluation Overview

Using OASIS, we wanted to answer two main research questions about notification management:

- How well can a notification management system detect and differentiate breakpoints within novel task sequences, i.e., sequences that lie outside of the data set originally used for training the statistical models?
- How does scheduling notifications at breakpoints affect users and their tasks? For example, does this reduce frustration or resumption lag, and how do users react to this type of scheduling behavior?

The first question is critical because to understand effects of scheduling notifications at breakpoints during authentic tasks, we need to understand how well the system can detect breakpoints. This is an important challenge since users' task behavior is known to be *highly* variable. It is thus not clear how well models for detecting breakpoints trained with one data set would perform on a different set.

The second question is important because effects of scheduling notifications have only been explored using controlled and relatively simple tasks where the moments were selected a priori (e.g., (Czerwinski, Cutrell et al. 2000; Adamczyk and Bailey 2004; Monk, Boehm-Davis et al. 2004; Iqbal and Bailey 2005; Bailey, Adamczyk et al. 2006; Iqbal and Bailey 2006)). It is thus unknown whether the previous findings generalize when notifications are scheduled during authentic, complex tasks and when the breakpoints are identified in real time.

8.2 Task Domains

Two domains were selected for this work; programming and diagram editing. These were selected because many users perform tasks in these domains, the tasks performed are typically complex, and the tasks are often intertwined with other activities such as Web browsing or managing communications. Microsoft's Visual Studio and Visio were

chosen as the specific applications for programming and diagram editing, respectively. For each application, we developed custom plug-ins that expose a large number of application events that can be monitored by our system.

8.3 Training Initial Models

To train the initial models, we used the model development component of Oasis. We trained a set of statistical models for detecting breakpoints when working within Visual Studio and Visio (Fine and Medium), and when switching between higher-level activities (Coarse).

8.3.1 Collection of Training Data

We recruited three users for each domain and Oasis was installed on their machine. Users were asked to run the model development component of Oasis the next time they would be focused on performing any task in their assigned domain (programming in Visual Studio or diagram editing in MS Visio) for at least 1-2 hours. They were also asked to perform the activity as they normally would during regular work, i.e., it was perfectly fine to check mail, play music, or read news intermittingly. Our software collected user's screen interaction, application and system-level events, and keyboard and mouse events. Users were compensated with $20 for participation.

The users performed a diverse set of complex tasks. For programming, one user was working on a Web-based graphics application using ASP.net. The second user was programming a notification display using Visual C#. The third user was writing C++ code to manipulate mouse and keyboard events for a distributed application. For diagram editing, one user was creating an information architecture for a research website. The second user was creating a project proposal outline for the local environment council. The third user was creating a system diagram for a poster.

8.3.2 Breakpoint Identification

Twelve independent observers were recruited for breakpoint identification. Using the breakpoint annotator tool in the model building component for Oasis, each observer reviewed two interaction videos and identified locations of perceived breakpoints and their type (fine, medium, coarse). The aggregated set of identified breakpoints was

filtered to include only those breakpoints for which there was a minimum threshold of agreement, following the procedure described in Chapter 6.

8.3.3 Strategies Used by Observers

Strategies to identify breakpoints were straightforward and consistent across observers. For example, Coarse breakpoints were stated to be switches to a new and unrelated application (but not switches back to Visual Studio or Visio), mostly email and IM clients. Medium was stated to be switches to related applications, such as browsing for references or finishing larger cognitive chunks within the same application. Fine was stated to be between smaller chunks of the main task (programming or diagram editing), e.g. after saving, after compiling or between repetitive actions within the same task.

Observers, however, did comment on the lack of information in the video regarding user inactivity. They could tell that there were no onscreen activities due to no movement of the mouse cursor, but it was difficult for them to decide whether this indicated that the user had taken a break or was just contemplating. Most of these inactive moments, therefore, were ignored by the observers as breakpoints; whereas the users did identify some of these moments as breakpoints, stating reasons such as someone dropping by their office and causing them to switch tasks – indicating the usage of contextual information not available to the observers.

8.3.4 Model Development

Based on the findings in Chapter 6, we set out to develop two classes of models – Global for predicting Coarse breakpoints, and Application-specific for predicting Medium and Fine. Training examples corresponding to each class was created taking into consideration associated features. For Global, features were more application independent where Application-Specific entailed features corresponding to that particular application.

We aggregated the training examples created from the true breakpoints and learned a single composite Global model, and two Application-specific models for Visual Studio and Visio. We followed standard techniques in model development: determining candidate features, generating training examples and learning the models.

Global Classifier: IB1 Instance Attribute Selection Algorithm: Wrapper
switchToMailEvent-1secBefore
switchToIMCLient-5secBefore
switchToOtherAppEvent-5secBefore
windowMinimizedEvent-10secBefore
switchToMailEvent-10secBefore
switchToEntertainmentAppEvent-10secBefore
switchToIMCLient-10secBefore
switchToWindowExplorer-10secBefore
switchToOtherAppEvent-10secBefore
highInactivityRate-10secBefore
unrelatedSwitches

Table 8.1: Predictive features for global model used to identify Coarse breakpoints

8.3.4.1 Determining Candidate Features

The first step in the model development was to determine features associated with breakpoints, which would be used to create training examples for each annotated breakpoint. The feature determination process is completely automated within the model development component in Oasis. The system logged a comprehensive set of application and system level events as the tasks were being performed, and each event was automatically coded as a candidate feature. Features were derived from these raw events as number of occurrences in different time windows prior to the breakpoint. For example, the system created a feature for the number of compilation actions in each of the following: 1 second, 2 seconds, 5 seconds and 10 seconds prior to the breakpoint. This allows us not only to associate immediately preceding actions with a breakpoint, but also consider a history of events that may contribute towards a breakpoint taking place.

The system also automatically generated a set of derived features from the raw keyboard and mouse events. For example, there were features in the different time

windows indicating rate of mouse movement, mouse direction change, rate of certain hot-key sequences etc.

The number of candidate features for the Global, Visual Studio and Visio breakpoint sets were 78, 335 and 444 respectively.

8.3.4.2 Generating Training Examples

The system automatically computed and aggregated the values corresponding to the candidate features for each breakpoint as a training example. However, since in practice a system will have to distinguish breakpoints from non-breakpoints (NABs), our training set would additionally require a sample of examples corresponding to NABs. For this purpose, the system divided the entire length of each video into 10 second bins, and any bin not having a breakpoint in it was automatically classified as a NAB, the corresponding training example being generated based on the end point of the bin.

Visual Studio Classifier: IB1 Instance Attribute Selection Algorithm: Wrapper	**Visio** Classifier: LogitBoost Attribute Selection Algorithm: Wrapper
windowActivated-5secBefore	highPasteRate-1secBefore
docSaved-10secBefore	eVisioIdle-2secBefore
windowActivated-10secBefore	eExitScope-5secBefore
projectConfigDone-10secBefore	eViewChangeBegin-5secBefore
buildDone-10secBefore	eViewChangeEnd-5secBefore
mouseMoves-10secBefore	eAppActivated-10secBefore
windowMinimizedEvent-1secBefore	eAppDeactivated-10secBefore
switchToSearchEngineEvent-1secBefore	eDocumentSaved-10secBefore
switchToOtherAppEvent-5secBefore	eBeforeShapeDelete-10secBefore
windowMovedEvent-10secBefore	eViewChangeEnd-10secBefore
switchToMailEvent-10secBefore	eShapeAdded-10secBefore
switchToOtherAppEvent-10secBefore	eMouseMoveBegin-10secBefore
unrelatedSwitches	switchToSearchEngineEvent-1secBefore
	switchToOtherAppEvent-2secBefore
	switchToWindowExplorer-5secBefore
	switchToOtherAppEvent-10secBefore
	userInactiveDuringBreakpoint

Table 8.2: Predictive features for the Visual Studio model and Visio model for predicting Medium and Fine breakpoints.

However, as expected, this resulted in a much larger proportion of NAB examples as compared to breakpoint examples. This would mean that the models learned would be biased towards making NAB predictions than breakpoint predictions. Since in practice we would want breakpoints to be identified with as much as certainty as possible, this bias would be in line with our intentions.

Training examples were separately generated for each class. For Global, the classification was between Coarse and Not Coarse, where any example other than a Coarse was annotated as Not Coarse. This included Medium and Fine examples as well as NABs, so that the learned model would be able to distinguish Coarse breakpoints from not only non-breakpoints, but other types of breakpoints too. For Application-specific,

Coarse examples were omitted from the training set, since in practice an application specific model would only see a case which has been identified as not being a Coarse breakpoint, hence there was no need to train the model on Coarse breakpoints.

Breakpoint examples were generated only for the true breakpoints. Remaining breakpoints that failed to make the cut were put in an 'ambiguous category' and bins corresponding to these ambiguous breakpoints were not considered while generating NAB examples. This step was taken to reduce the noise in the training data. We made the tacit assumption that since some observer had identified that moment to be a breakpoint, there may be something salient about that moment that caused it to be considered so, even if it was not widely agreed upon. Because of this apparent ambiguity these moments were not considered as NABs either.

To create a single model for each application, we aggregated training examples across all three tasks in each domain. Similarly, to create an all-encompassing application independent Global model, training examples across the six tasks were aggregated. The total number of training examples for the Visual Studio model and the Visio model was 1105 and 790 respectively, and 2105 for the Global model.

The general model for detecting Coarse breakpoints had 11 features (predictive events), see Table 8.1. Application specific models for detecting Medium and Fine within Visual Studio and Visio had 13 and 17, respectively, as shown in Table 8.2.

8.4 Training Results

Table 8.3 and Table 8.4 show the accuracy of the models, with the diagonals showing correct predictions. Accuracies (percentage of correct predictions) were high (> 87%). This is due in part to the large number of NABs, and the models having predicted most of them correctly (99% for the general model, 98% for programming, and 97% for diagram editing).

To evaluate how well the models predict actual breakpoints, we computed Recall (percent of true breakpoints correctly predicted) for each breakpoint type. Recall of Coarse was 71% (60/85). Recall of Medium was 84% for programming and 56% for diagram editing. Recall of Fine was 96% for programming and 58% for diagram editing. The models missed some breakpoints. This means that occasionally opportunities for scheduling notifications at breakpoints would be missed. However, the most egregious error, wrongly predicting a moment to be a breakpoint when it is not, was very low (<0.5%).

Recall values showed that our models are able to detect and differentiate breakpoints in the *training data* with reasonable accuracy. Results were consistent with results reported in Chapter 6. This gave us high confidence that these models were robust enough for

		Predicted	
		Coarse	Not Coarse
Actual	Coarse	60	25
	Not Coarse	19	2188

Table 8.3: Predicted vs. Actual for the general model used to detect Coarse breakpoints. Overall accuracy was 97.97%.

		Predicted		
		Med	Fine	NAB
Actual	Med	64, 40	0, 12	12, 20
	Fine	0, 0	104, 93	4, 67
	NAB	16, 5	9, 16	1034, 711

Table 8.4: Predicted vs. Actual for the application models used to detect Medium and Fine for (programming, diagram editing). Overall accuracies were 96.7% and 87.6%, respectively.

testing on novel task sequences.

8.5 Study 1: Evaluate Breakpoint Detection

The purpose of the first study was to evaluate how well breakpoints could be detected and differentiated within novel task sequences, i.e., sequences generated outside the data originally used for training and evaluation.

8.5.1 Users and Tasks

Six users (3 per domain) were recruited for the study. Each user reported having moderate to expert skills in the respective domain. We installed Oasis on the user's machines, allowing them to work on their own tasks and in their own environments. Users were asked to run the model development component of Oasis the next time that they would be focused on performing their task (with the necessary application) for an hour or more. Users were informed that they should maintain their normal work practice, i.e., switch applications, chat with others, or browse the Web, as desired.

For programming, one user was developing a graphical interface for a mobile device using Visual C#. Another user was programming a graphics rendering tool using Visual C++. The third user was writing code to process image files in Visual C#. For diagram editing, one user was creating an outline for her doctoral thesis. The second user was diagramming the logic flow of an interactive game. The third was creating a process diagram for a research paper.

8.5.2 Procedure

The procedure consisted of two phases. In the first phase, users worked on their selected tasks with our software running. Our system monitored the event stream and used the originally trained models to detect whether and what type of breakpoint had occurred. Event data was pooled in 3 second bins, as this was found in the training phase to give the best accuracy. Each bin of event data and the related prediction (Fine, Medium, Coarse, or NAB) were logged to a file. Users' screen interaction was recorded, and could be synchronized with the event data.

For the second phase, users used a software tool to review their own interaction videos and identify locations of the breakpoints and their type. We asked the users themselves to

identify the breakpoints, rather than utilize independent observers, because a system like ours will ultimately need to be evaluated based on how well its predictions match a user's own understanding of their tasks. Note that using observers in the training phase was important because it allows the most perceptually salient breakpoints to be detected and used for training, resulting in robust models.

8.5.3 Measurements and Analysis

We compared the breakpoints detected by the composite models to the breakpoints identified by the users. A system-identified breakpoint was considered a match with a user-identified breakpoint if they were within 10s and of the same type. After testing several values, a 10s window seemed to best compensate for the difference between when a breakpoint occurred and when a user annotated it.

8.5.4 Results

Table 8.5 and Table 8.6 show the distribution of system- vs. user-identified breakpoints, with the diagonals showing the matches. Data was aggregated across users in each

		System-identified			
		Coarse	Medium	Fine	NAB
User-identified	Coarse	33	12	0	35
	Medium	22	5	1	22
	Fine	25	1	1	29
	NAB	68	7	0	3299

Table 8.5: System- vs. User-identified breakpoints for tasks in diagram editing. Overall accuracy was 93.8%.

domain. The overall accuracies were high (> 90%), but again, this is due in part to the large number of NABs, and the ability of the models to correctly predict most of them.

Recall of Coarse was 41.5% and 41.3% for programming and diagramming, respectively. Recall of Medium was 20.4% for programming and 10% for diagram editing. Recall of Fine was 15% for programming and only 1.7% for diagram editing. Admittedly, these results were much lower than expected given the training performance.

Closer inspection reveals that the majority of mismatches was due to users identifying breakpoints as Medium or Fine, while the system identified those same breakpoints as Coarse. On the one hand, this is in fact a very positive result because it shows that users and the system were agreeing on the *location* of the breakpoints, but were disagreeing on the *type* of those breakpoints.

Part of the reason behind the low accuracy in identifying the type of breakpoint was the inability of the models to understand users' task context. For example, one user switched repeatedly between Gmail and Visio to retrieve documents related to her task. The system identified these switches as Coarse breakpoints, while the user identified them as Medium or Fine considering the relevance to the ongoing task. This illustrates how task context influences perceptions of breakpoint type as well as the necessity and challenge of integrating such context into the models.

The most egregious type of error, detecting a breakpoint when none was present, was still very low; 2.8% for programming and 2.3% for diagramming.

8.5.5 Discussion

Results from this study highlight the significant challenge of using composite models to detect and differentiate breakpoints within novel task sequences. Several methods could be pursued to increase the accuracy of such models. For example, models could be

		System-identified			
		Coarse	Medium	Fine	NAB
User-identified	Coarse	44	17	2	43
	Medium	40	21	1	41
	Fine	35	17	18	49
	NAB	69	12	8	3092

Table 8.6: System- vs. User-identified breakpoints for programming. Overall accuracy was 90.5%.

trained using a much larger data set, they could be trained on a per user basis, or a combined approach could be followed. Including features representing additional task context must also be pursued.

Yet even if this problem can be mostly solved, and we believe that it will given the active ongoing research in this direction (Horvitz, Koch et al. 2002; Horvitz, Kadie et al. 2003; Fogarty and Hudson 2007), a critical question still remains. How would scheduling notifications at breakpoints (assuming correct identification) affect users and their tasks?

To provide a first answer to this question, we wanted to build upon the fact that the models were able to identify the location of breakpoints with reasonable accuracy. We thus retrained our models to identify moments as either breakpoints or NABs, i.e., we collapsed breakpoints into one type. Applying these new models to the same data set resulted in 59% and 52% of user-identified breakpoints being correctly identified for programming and diagram editing, respectively. We judged this to be sufficient for moving forward with our second study.

As part of the study, we also wanted to test the effects of scheduling notifications at each type of breakpoint. We decided to use our system (with the new models) to *detect* the locations of breakpoints and to ask the users to identify type. This would effectively compensate for the differentiation performance of the models, and allow breakpoint type to be included in the analysis. We felt that this was important because it would show whether the ability to differentiate breakpoint type for scheduling notifications would have any benefit for users in practice.

8.6 Study 2: Evaluate Effects of Scheduling Notifications

The purpose of the second study was to evaluate how scheduling notifications at breakpoints impacts users and their tasks. We also wanted to investigate the interaction between notification content and scheduling policy. We thus designed notifications to be either relevant to the task or of general interest to the user (but not task relevant).

8.6.1 Users and Tasks

16 users (8 per domain) were recruited for the study. Users reported having moderate to expert skills in the respective domain. Users received $50 for participating.

Tasks required users to develop solutions to challenging, ill-structured problems during the study. Only high-level descriptions were provided, and it was up to the users to work out a desired solution. For programming, the task was to create a user interface for applying convolution filters to images, and to implement at least three filters. Users were provided with a description of convolution filters, pointers to Web-based resources, and a skeleton C# project that they could build upon, if desired. Users were asked to make their code as efficient and readable as possible. MS Visual Studio was used for the task.

For diagram editing, the task was to design a floor plan for a model work space in a Computer Science building. The space had to accommodate 6 students, and needed to include cubicles for daily work, a joint lab for conducting experiments, and a service room for relaxing and eating. Users were asked to create as many design alternatives as possible. The task was performed using Visio.

For both tasks, users were free to browse for examples, references, or any other desired information online. Users were informed that they needed to spend 2 hours on the task and should work at their own pace. Given the length of the task, users were also informed that they were free to perform other personal tasks such as check mail or read their favorite online news site. The goal was to have users work in a manner similar to how they would in practice. To facilitate motivation, an additional $50 was offered to the user who created the highest quality solution in each task domain, as judged by independent experts.

OASIS was installed on the experimental machine and monitored the user's activity to detect breakpoints. It also managed notification requests from a custom application.

8.6.2 Notifications

As users performed tasks, they occasionally received notifications. Two types of notifications were used:

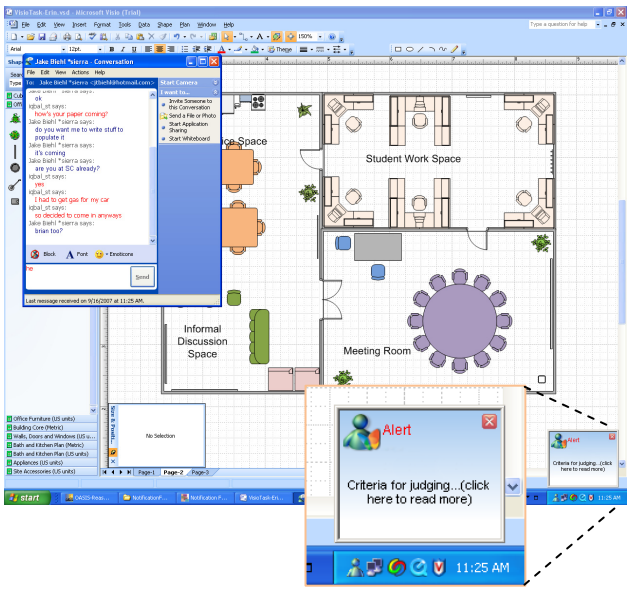


Figure 8.1: Screenshot of a notification arriving as the user transitioned from diagramming to a secondary task of chat. Since a notification was pending, and the system detected this moment as a breakpoint, the notification was delivered.

- *Relevant*. These notifications provided examples (e.g., source code or floor plans), useful tips, or additional criteria that would be used for judging solution quality.
- *General Interest*. These notifications presented recent news grabbed from Google News or announcements from our department's or institution's homepage.

Two policies were used for delivering notifications:

- *Defer to breakpoint*. A request would be sent to OASIS, which would schedule the notification to appear at the next breakpoint detected in the user's task sequence.
- *Immediate*. The notification was delivered immediately.

16 notifications were generated randomly within intervals spanning the 2 hour task period by a custom application. 16 was based on prior work showing that computer users receive on average about eight notifications per hour (Iqbal and Horvitz 2007).

4 notifications appeared as soon as they were generated (Immediate). The other 12 were to be scheduled by OASIS to appear at breakpoints. However, depending on our system's detection of breakpoints, it was possible for users to receive fewer. We chose to have more scheduled (12) than immediate (4) notifications to try to ensure that each type of breakpoint would be used (how types were identified is discussed below). Half of the notifications were Relevant and the other half were of General Interest. These were balanced between the two scheduling policies.

Notifications were rendered as a non-modal window in the lower right of the screen and contained a short snippet of the message content (Figure 8.1). The window persisted for 7s. Users could select the text snippet to read the full message. The overall design was meant to simulate a technique commonly used today (e.g., by MS Windows).

8.6.3 Experimental Design

We used a 2 Activity (programming, diagram editing) X 2 Policy (breakpoint, immediate) X 2 Content (relevant, general interest) mixed design. Policy and Content were within subjects while Activity was between subjects.

8.6.4 Procedure

Upon arrival at the lab, we went through an informed consent process with the user. The user was provided with a description of the task and allowed to ask any questions. Users were contacted prior to their scheduled session so that the local machine could be configured with their favorite applications and bookmarks as best as possible.

The user was informed that during the task, notifications containing relevant or potentially useful information would occasionally appear. They were asked to select the notification whenever they noticed it and/or the task allowed. If selected, a dialogue box would immediately open, asking the user to rate his or her frustration with having received the notification at that moment. Once the rating was made, a Web page opened showing the full content of the notification. Users were then free to proceed as desired. The task session lasted for 2 hours.

Afterward, a post experiment interview was conducted. The experimenter launched a tool that showed the user's interaction video along with the locations of the system-identified breakpoints. The experimenter navigated to each breakpoint and asked the user to agree or disagree with whether that moment was a breakpoint. If agreed, the user identified its type (coarse, medium, or fine) based on given descriptions. If disagreed, the user scrubbed the video to identify the closest point where they would have preferred to receive the notification and explain why.

8.6.5 Measurements

The following measurements were taken:

- *Frustration*. The rating was made using a 7-point Likert scale, ranging from very pleasing to very frustrating.
- *Reaction time.* This was measured as the time between when a notification appeared and the user selected it.
- *Resumption time.* This was the time from when the user closed the notification content page to when focus on suspended activity was resumed. This time would also include diversions into other activities, if any.

These metrics have been used to measure the effects of interruption in many previous studies (e.g., (Iqbal and Bailey 2005; Bailey and Konstan 2006; Iqbal and Horvitz 2007)). In addition, we solicited feedback on how users felt about having notifications deferred until breakpoints. We also analyzed interaction videos to compare how users responded to notifications delivered under different scheduling policies.

8.6.6 Results

Out of a maximum possible 256 notifications, 64 were to be delivered under Immediate and 192 were to be delivered under Scheduled. Out of the 64 Immediate, 1 was not delivered due to the user finishing the task before the notification could be generated.

Out of the 192 Scheduled, 170 were delivered. 109 of these were delivered at moments that users agreed were breakpoints, resulting in 64% precision (percentage of predicted moments that were actually breakpoints). Of these 109, users identified 26 as Coarse, 44 as Medium and 39 as Fine. These user-specified types were used as the values for the Policy factor in our analysis. The 61 cases where there was no agreement were excluded, since they are similar to the data collected for Immediate.

The remaining 22 Scheduled could not be delivered due to the system not being able to detect any breakpoint between when the notifications were generated and the end of the task session. These were excluded. Also, in some cases, a notification appeared while the user was still responding to another. This resulted in an additional 29 notifications being excluded to avoid any potential confounding effects. In sum, our rigorous filtering process left us with 143 data points for analysis.

For Scheduled notifications that were delivered, the mean deferral time (the time from when they were generated to when they were delivered) was 88.6s (S.D. 139.3s). Table

8.7 shows a breakdown across the different types of breakpoints. These times are consistent with breakpoint distances reported in Chapter 6, and with transition times to non-busy states in (Horvitz, Apacible et al. 2005).

Breakpoint Type	Mean Deferral Time(sec)	Standard Deviation
Coarse	127.69	146.61
Medium	107.49	159.78
Fine	51.6	85.65

Table 8.7: Mean deferral time in seconds across different breakpoint types.

8.6.7 User Reactions and Behavioral Responses

The concept of scheduling notifications at breakpoints was well received by users and matched what they themselves preferred. For example, when scrubbing the video (see Figure 8.2) to select preferred moments to receive notifications that had not appeared at a breakpoint, they almost always described a moment that indicated the completion of an action. This is exemplified in many of their explanations:

"After I have added this room [would have been a good moment]"

"I wish it had waited until I was done with this [the stairs]"

"Just before that - where I scrolled down the window …just when I ended this method."

"I would have preferred it [the notification] when I had just finished this line."

We also discovered that there was an interaction effect between scheduling policy and notification content. For example, users expressed wanting notifications relevant to their ongoing activity to be delivered at Medium or Fine breakpoints as opposed to Coarse. Even though this may cause higher localized costs (e.g., in terms of resumption lag (Iqbal and Bailey 2006) or reaction time), users perceive a larger global benefit because the notification is received when its content can be best utilized, and precludes the need for a context switch. When a relevant notification was delivered at Coarse, we often observed users immediately returning to the activity they had just left. Users expressed they disliked these occurrences since they were intending to move away from the ongoing task. Receiving a relevant notification caused them to abandon that task switch.

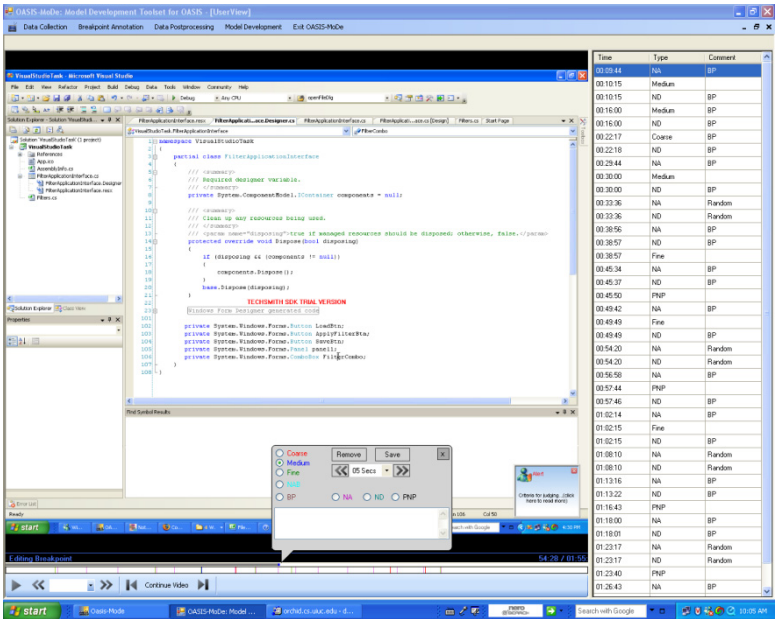


Figure 8.2: A screenshot of a user scrubbing the video to validate breakpoints identified by the system. The users could scrub back and forth to select a better moment and also specify the type of breakpoint, if any.

For General Interest notifications, users stated that they wanted them to be delivered only at Coarse. This was also evident in their task behavior. For example, if a general interest notification appeared at Medium or Fine, users cursorily read the content and attempted to return to the suspended task as soon as possible. If it appeared at Coarse, users would often read the content in its entirety, and then proceed with their intended task switch.

The key implication of these results is that notifications deemed relevant to the ongoing activity should be scheduled at Medium or Fine, while notifications of general interest should be scheduled at Coarse. This also indicates that notification scheduling systems must be able to detect all three types of breakpoints in practice.

A related but less commonly observed behavior was users initiating *chains of diversion* (Iqbal and Horvitz 2007). This refers to the activities that a user performs after having attended to a notification but before resuming their suspended task.

25 chains of diversion were observed, 11 for diagram editing and 14 for programming. The nature of the diversion was a function of both the ongoing task and the notification content. For example, during diagram editing, *general interest* notifications caused users to enter a chain of diversion most often (9 of 11). During these diversions, users would

check mail, weather or movie schedules, or browse online news. For programming, users went on a chain of diversion most often (11 of 14) after having received a *relevant* notification. Diverted activities were often related to the programming task, e.g., looking up code samples or browsing online forums. Policy did not seem to have an impact on the chain of diversion.

8.6.8 Frustration

A 3-way ANOVA showed main effects of Content ($F(1, 109)=13.9$, $p<0.001$) and Policy ($F(3, 109)=5.4$, $p<0.002$), and an interaction effect between Policy and Activity ($F(3, 109)=4.7$, $p<0.004$).

For Content, notifications that were of general interest caused more frustration ($\mu=4.98$, S.D.=1.81) than those that were relevant ($\mu=3.59$, S.D.=1.71). Several users stated that even if a notification may have been initially perceived as disruptive, if they determined the content to be relevant, they were more tolerant towards it. This finding is consistent with results in prior work (Czerwinski, Cutrell et al. 2000; Gluck, Bunt et al. 2007).

Due to the interaction, we examined effects of Policy within each Activity separately. For diagram editing, Policy had a main effect on frustration ($F(3, 52)=6.2$, $p<0.001$). Post hoc tests showed that notifications delivered at Coarse ($\mu_C=3.6$, S.D.=1.99) caused lower frustration than at Fine ($\mu_F=5.5$, S.D.=1.7). Notifications delivered at Medium ($\mu_M=2.6$, S.D.=1.6) caused lower frustration than at Fine ($p<0.001$) and Immediate ($\mu_I=4.5$, S.D.=1.58; $p<0.037$). No other differences were found.

For programming, trends were in the expected direction ($\mu_C=3.28$, $\mu_M=4.39$, $\mu_F=4.33$; $\mu_I=4.82$), but did not reach a level of significance. The lack of effect may be due to the fact that programming induced higher cognitive demands than diagram editing, causing users to experience similar levels of frustration across policies. This is further supported by the fact that 15 of the 16 notifications that users failed to respond to were during programming.

8.6.9 Reaction Time

A 3-way ANOVA did not reveal main effects of the factors on reaction time. However, inspection of the graph showed a very salient pattern in how users were reacting to relevant notifications delivered at breakpoints versus immediate (see Figure 8.3). To

explore this further, we collapsed breakpoints into a single Breakpoint level and reran the ANOVA for only relevant notifications.

Results showed a main effect of Policy (Breakpoint, Immediate) on reaction time ($F(1, 66)=3.78$, $p<0.056$). Users reacted to notifications at Breakpoints ($\mu=3.07$s, S.D.=1.2) faster than at Immediate ($\mu_I=4.08$s, S.D.=3.13).

A plausible explanation for this outcome is that for Immediate, users needed to first externalize information (e.g., finish the current line of code or complete alignment of shapes) into the task environment, thus causing slower reaction time. When delivered at breakpoints, users could switch their attention more readily to the notification, resulting in faster reaction time. Analysis of user behavior in the videos confirmed the veracity of this explanation.

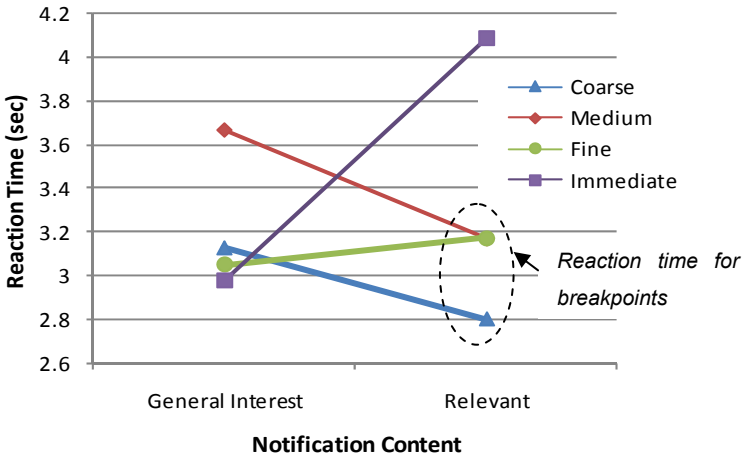


Figure 8.3: Effects of Policy on reaction time. For Relevant, reaction times were faster for notifications scheduled at breakpoints compared to those delivered immediately.

The same pattern was not found for notifications that were of general interest. This may be because users did not anticipate being away from the task for long, therefore were less concerned about externalizing information.

8.6.10 Resumption Time

An ANOVA revealed an interaction between Content and Activity for resumption time ($F(1,109)=7.75$, $p<0.006$). For Diagram Editing, users resumed their activity faster after responding to notifications that were relevant ($\mu=4.65$s, S.D.=4.4s) compared to those

that were of general interest (μ=23.1s, S.D.=41.4s; $F(1,54)=5.91$, $p<0.018$). This result can be attributed to users initiating more chains of diversion after receiving general interest notifications, as previously discussed (see User Reaction).

For programming, users resumed their activity faster after responding to notifications that were of general interest (μ=8.8, S.D.=21.1) compared to those that were relevant (μ=16.6, S.D.=21.5). Differences were due to users having more chains of diversions after receiving relevant notifications, though these did not reach significance. Overall, these quantitative results reflect the qualitative observations discussed under User Reactions.

8.7 Discussion

A central goal of this study was to evaluate the impact of using a notification management system to schedule notifications on users and their tasks. Our results showed that users experience meaningfully lower frustration when notifications are scheduled to occur at breakpoints than when delivered immediately. Scheduling notifications at Coarse and Medium results in lower frustration than when scheduled at Fine. An explanation is that users may experience temporary reduction in memory load at these moments, or are at a transition in their action sequence. Our results on frustration are consistent with (Iqbal and Bailey 2005).

Users reacted faster to notifications that were scheduled at breakpoints. For notifications delivered immediately, users would typically externalize their current thought into the task environment or complete their current action before responding. Similar observations have been noted in (Fogarty, Ko et al. 2005; Iqbal and Horvitz 2007). Interestingly, this behavior was not observed when notifications were scheduled at breakpoints. This is likely due to users having just completed their current thought or action at that moment. This difference in user reaction was most apparent for relevant notifications, and seldom surfaced for general interest notifications. One explanation stems from the fact that users would quickly dismiss general interest notifications, regardless of how they were scheduled. Perhaps because users anticipated only being away from the task for a short time, they did not perceive the need to externalize their current thought.

Reductions in frustration and reaction time must be balanced against the time that notifications are deferred. Our results show that the average deferral time was less than 90s. We believe this provides an acceptable balance.

Our results did not show that scheduling notifications at breakpoints affects users' resumption time. Results did show, however, that resumption time depends upon the relevance of a notification to the user's ongoing activity. This is often due to users following *chains of diversion*. For example, for diagram editing, notifications of general interest caused users to initiate chains of diversion most often, whereas for programming, it was the relevant notifications that caused these diversions. One implication is that task reminder tools (e.g., those discussed in (Iqbal and Horvitz 2007)) can use the relevance of a notification to help detect whether a user is following a chain of diversion or not.

Another important finding is that users prefer having notifications scheduled at breakpoints. The reason is that this technique closely reflects their own preference for how notifications should be managed. For example, when retrospectively selecting preferred moments for receiving notifications, users identified moments that represented the end of an action corresponding to the completion of a cognitive chunk, e.g., the end of a series of code edits. This strongly indicates that users would accept systems that schedule notifications at breakpoints in practice.

Our results provide further insights into how applications should utilize defer-to-breakpoint policies. For example, applications that generate notifications that are relevant to the user's ongoing activity should request that they be delivered at Medium or Fine breakpoints. This would allow notifications to be delivered when they have the most utility (i.e., *during* the activity), but at less disruptive moments. In contrast, for notifications of general interest, applications should request that they be delivered at Coarse breakpoints. These would be the moments when delivering such notifications would be least disruptive. Urgency of notification content should also be considered when selecting an appropriate policy and timeframe (Horvitz, Apacible et al. 2005).

A second goal of this work was to test the performance of our system in detecting and differentiating breakpoints within novel task sequences. In Study 1, results showed that using composite statistical models in the system can detect breakpoints with 52% (diagram editing) and 59% (programming) recall for novel task sequences. In Study 2,

using the models resulted in a precision of 64%. These are promising results since the breakpoints were the ones that users themselves identified in their own tasks. This is an important outcome as it shows that composite models can detect breakpoints for different users performing the same type of complex task with reasonable accuracy.

However, the models performed poorly for differentiating breakpoint *type*. For example, in Study 1, the models differentiated breakpoints with only 2-42% accuracy. Applying the models to differentiate breakpoints in Study 2 did not yield better results. One implication is that systems may want to use such models only for *detecting* breakpoints, i.e., without differentiation. This can be done with modest accuracy, but our results show that users can benefit in terms of reduced frustration and reaction time.

More flexible scheduling policies can be offered if the type of breakpoint could be differentiated. These policies would be useful, e.g., to allow notifications to be more effectively scheduled based on their relevance. Having various policies available would also allow applications to choose an appropriate balance between notification timeliness and costs of interruption for the user. Further work is needed to understand how to improve models for differentiating breakpoints. Related work on modeling interruptibility may provide applicable insights (Horvitz and Apacible 2003; Fogarty, Hudson et al. 2004; Fogarty, Ko et al. 2005).

Two complex task domains were used for this research – diagram editing and programming. Both domains require some form of content generation. Our results are thus most applicable to domains with similar characteristics, e.g., document editing, image manipulation and electronic communication. Future work should study the effects of notification scheduling within other types of task domains such as information-seeking and data manipulation.

CHAPTER 9

Discussion and Future Work

In previous chapters we presented a solution to the ensuing problem of interruption management for the desktop interface. In this chapter we discuss some of the broader issues related to our solution in context of the domain of interruption management. We also discuss future directions with this work.

9.1 Effectiveness of Notification Scheduling

A central goal of this dissertation was to gain a deeper understanding of the effectiveness of automated notification scheduling for users. Given that underlying breakpoint models can be trained to provide improved levels of accuracy, our evaluations showed that there are tangible benefits of automated notification scheduling both in terms of qualitative and quantitative effects. For example, the notion of deferring notifications until breakpoints seemed to match how users themselves preferred to deal with notifications, indicating that users will be willing to adopt such systems in practice. An interesting and unexpected discovery, however, was that users were willing to accept higher local cost in order to improve global benefit. As a result they preferred relevant notifications to occur at relatively costlier medium and fine breakpoints. This suggests that users could perceive the need for striking a balance between cost and benefit of a notification and wanted to adjust accordingly.

Quantitatively our results showed promising trends. For example, notifications delivered at breakpoints had faster reaction times than notification delivered at non-breakpoint moments. However, the difference was only about a second, which even though statistically significant, may not seem meaningful. We argue that even though the gain in reaction time through scheduling a single notification may not make a big difference, extrapolating the gain over a longer period of time, over many notifications

and many users will have a cumulative effect, having a substantial practical and positive impact for users.

In addition to providing initial evidence that notification scheduling will have tangible benefits for the end user, our results also demonstrated complex relationships among demands from ongoing task and relevance of notification. For example, during diagram editing, *general interest* notifications often caused users to start chains of diversion to *unrelated* actions. On the other hand, for the more cognitively demanding programming task, it was a *relevant* notification that would result in users start a diversion chain to *related* actions. This may be due to the varying demands placed by the ongoing tasks on users, thereby affecting their willingness to switch tasks. There appeared to be an effect of the relevance of the notification that also contributed to how users reacted to a notification for different types of tasks.

Such observations provide early evidence that notification scheduling is beneficial to the end user, and additionally, highlight the challenges in realizing these policies in practice. For example, to support notification scheduling for a situation similar to above, an underlying system will have to be aware of the type of task, the content of the notification and its value to the user or the ongoing task, and then make the appropriate decision. Also through our findings, we have demonstrated that to realize benefits of scheduling in practice, there is a need to focus on developing more effective models for breakpoints, which will only improve performance. Mechanisms to predict relevance of a notification to an ongoing task also need to be in place, as our work shows this to be an important factor in deciding when to interrupt the user. Availability of our framework provides a test bed to test new models and policies in authentic settings, which was until now missing from the interruption literature.

9.2 Improving Performance

Part of the effectiveness of our system depends on the system's ability to pick out breakpoints accurately. Breakpoints are detected and differentiated in real time using models developed a priori. An important question is how these models should be developed in order to ensure better performance.

In our system, we used composite models created from a small set of representative users. These models were validated on a different group of users to evaluate how generalizable the models were. Although the accuracies of models on the training data were good, the accuracies when applied on a different set of users had room for improvement, especially in differentiating breakpoints. We proposed several approaches to address this issue in Chapter 8, e.g. creating composite models from a wider variety of users, creating personalized models to capture individual behavioral differences etc. Our system provides an interface for creating these models that can be directly used by the reasoning system, making it easier for those who wish to create their own models.

However, the larger issue lies with the fact that to ensure that users will continue to use the system, there is a need to demonstrate initial benefit to the user *without* requiring users to put in a larger effort in customizing the models. This necessitates providing default models with the system that perform reasonably well. Performance of composite models can be improved if the models are built from a large sample of data from a wide variety of users. This opens up a new challenge in collecting, organizing and sharing data that can be used to develop these kinds of default models, potentially benefitting many users and researchers.

Providing interfaces for customization of models also presents an interesting challenge. Ongoing work (Stumpf, Rajaram et al. 2007; Tullio, Dey et al. 2007) has been exploring feasibility of online learning of user activities where the user provides feedback about decisions made by machine learning algorithms. This may be a fruitful and effective way to fine tune models used within intelligent systems such as OASIS.

9.3 Effects of Model Performance on Notification Scheduling

As research on improving model performances continues, a valid question arises in terms of how false positive and false negatives in breakpoint identification affect notification scheduling. Throughout this dissertation our focus has been on keeping false positives low, since these indicate moments wrongly identified as breakpoints. Delivering a notification at a non-breakpoint moment essentially replicates how notifications are delivered in today's interfaces. While this may be allowable and even inevitable in some cases due to lack of decisive information, the percentage of false positives should be kept

low. With high percentages of false positives, users will experience the same disruption as they do without scheduling, the value of scheduling will be diminished and users will fail to benefit from using the system.

On the other hand, false negatives occur when true breakpoints are missed by the system, potentially missing opportunities for notification delivery. This may impact the purported benefits of scheduling, as notifications may be delayed for too long while the system waits for a suitable breakpoint to occur. One way to address this problem is lower the threshold that determines whether a moment is a breakpoint or not. This will allow more 'on-the-fringe' breakpoints to be identified as breakpoints by the system, thereby reducing the probability that notifications will have to be delayed for longer periods of time.

However, while lowering the threshold may reduce the false negatives, it can potentially increase the number of false positives. This presents an interesting dilemma as this yet again demonstrates the difficulty in striking a desired balance between costs and benefits. We suggest addressing this on a case by case basis, decided by the users a priori based on the type of task. For example, in situations where reducing cost is important (e.g., when the ongoing task is safety critical and/or has a higher priority), the threshold could be increased. In situations where the incoming notification is more time critical, the threshold could be reduced, thereby allowing more breakpoints to pass through. However, while this approach may temporarily address this issue, it is not a panacea and eventually these systems will require models that have high enough accuracies to begin with.

9.4 Large Scale Deployment of the System

The goal of this dissertation was to demonstrate the effectiveness of automated notification scheduling, where notifications are delivered at breakpoints using an automated system. For demonstration purposes, the system supported two common domains – programming and diagram editing, where applications sent back events to the system which in turn are evaluated using models of breakpoints. In order to deploy the system as a fully functional prototype for managing notifications, the immediate next step is to instrument most, if not all applications commonly used in the desktop domain to

send back events to our system. This will allow our system to have a wider coverage in determining opportune breakpoints at which to schedule notifications.

There are three major requirements within system development that are required to make this happen. First, applications will need to expose events that can be coded within custom plug-ins to report back to the underlying notification management system, similar to the plug-ins built for Visual Studio and Visio in our framework. Application developers may consider making APIs available for access to the application level events that can be coded into plug-ins with relative ease, especially if the application is envisioned to be used frequently.

Second, communication channels need to be set up between applications and the notification management system. This will allow application level events to be sent to the system, as well as notification requests from applications wishing to interrupt users with information. In addition, the system needs to be able to send clearance flags from the system to notifying applications to allow notifications to go through. Our current implementation supports this through asynchronous callbacks and could be extended to support many more applications. As events are sent to the system only as they occur (rather than polling), we do not anticipate the system will consume too many computing resources.

Finally, our system requires models of breakpoints to be developed at both the global level (for coarse) and at the application level (for medium and fine). Depending on a user's expected level of usage, models could potentially be developed only for the most commonly used and most critical applications to the users. As Medium and Fine breakpoints indicate opportune moments within a context of a task, models should be developed for a particular task/application only if notifications are expected to be urgent and/or relevant to the ongoing task. For example, if a user mostly performs coding tasks within Visual Studio and switches back and forth between Firefox, Visual Studio and Word; and receives notifications that are often related to his coding task, then it would be prudent to develop application level models for Visual Studio, Word and Firefox. Whereas, if Photoshop is used only occasionally, then perhaps the effort required to develop such models for Photoshop is not justified. However, with ongoing efforts on

model development, we envision the effort to develop such models will be eventually reduced and creating models on the fly may not be too difficult to achieve.

Our system currently exists on the same machine, but as we are using the TCP/IP protocol for communication, it could be set up on a central server to manage notification delivery remotely. However, how this would play out with network delays, especially with breakpoints being rather ephemeral is not clear. We intend to investigate this in future.

9.5 Other Factors to Consider While Delivering Notifications

This dissertation primarily focused on one aspect of notification management, intelligently manipulating timing of notification delivery. The goal was to reduce interruption costs so that users are less disrupted while ensuring that notifications are delivered on time. Our approach considered one important determinant of interruption cost – task activities of the users in the desktop interface. While this provides an approximation of a user's task engagement, it does not capture other rich information about the user's tasks available through knowledge about the user's current social context, engagement in other activities away from the computer or experiences in recent past. For example, interruption cost should also factor in social information such as whether the user is on the phone or conversing with someone in their office. Vision or speech recognition systems could be used to detect social cues and determine the importance of the social engagement and potential cost of an interruption at that moment. Vision and sensor based systems could be used to learn models of user engagement outside the desktop interface, using features such as artifacts being used (e.g. whiteboards, paper), user posture (e.g. reclining, hunched forward), location in room etc. Similar environmental features have been used to inform models of interruptibility with up to 88% accuracy (Fogarty, Hudson et al. 2004).

Ultimately, a computer user's interruptibility in the workplace is a function of task level engagement on the computer, task level engagement outside the computer and social engagement, among other factors. While this dissertation only focuses on task engagement on the computer, it demonstrates the positive impact of this single dimension

on the interruption cost and highlights the need to combine information regarding other sources of engagement.

Another important factor contributing to the cost is how the notification is presented to the user. Even if we solve the timing problem, the presentation of the notification can impact how well information is conveyed to the user. For example, determining the appropriate modality of conveyance (e.g., visual or auditory) may depend on how salient the notification should be, often a function of its urgency. Also, choice of modality should take into consideration the user's current level of engagement and possible conflicts in sensory channels that are already engaged (Latorella 1998).

Given that the modality of the notification is determined, selecting values of the parameters presents a challenging problem. For example, for an auditory alert, the pitch, tone or frequency are known to impact how humans perceive alarms in high-stress situations (Stanton 1994). In the desktop interface, where notifications have varying degree of importance to the user based on the current context, the balance is dynamic, and often impacted by the relative importance of the notification to the user. The choice of the parameters therefore becomes much more important. Similarly, for a visual presentation, the organization (Van Dantzich, Robbins et al. 2002), spatial location (Osgood, Boff et al. 1998), visual effects (Maglio, Barrett et al. 2000) or attentional draw (Gluck, Bunt et al. 2007), and mapping these factors to the utility of the notification can play a major role in how the notification is perceived by the user.

The SEEV model, proposed by Wickens et al. (Wickens, Helleberg et al. 2001) may provide guidance on designing notifications and predicting attention on different presentation styles. While this model was primarily developed to predict scanning behavior during visual attention tasks, principles of the model could apply to other modalities also.

9.6 Generalizability of Approach

The notion of leveraging breakpoints as opportune moments for interruption works for goal directed desktop tasks. The underlying theory can be generalized to goal-directed tasks in other domains also. For example, in aviation, a cockpit checklist task could be potentially decomposed hierarchically into its components subgoals. Breakpoints at

multiple levels could be determined in the hierarchical goal structure. Similarly, breakpoints can be detected within goal directed physical tasks (between riding a bike and parking it) and notifications could be presented at those moments.

For continuous attention tasks, e.g. driving, the notion of breakpoints may need to be adjusted slightly. For these cases, being able to determine the attentional demands may prove to be more informative for notification management systems. As present day vehicles come equipped with many devices, information being transmitted to and from the vehicle could be used to gather relevant information. For example, knowing when the user is at a stoplight may be useful in delivering a notification. This can be easily achieved through GPS systems used for navigation. Similarly, knowing the user's internal environment (e.g. whether the user is on the phone, playing music, navigating through GPS system – interruption management systems can infer a user's workload and manage interruptions accordingly. Breakpoints, therefore in these cases, is not necessarily a break in the task flow, but a break in the attentional demands. Further research is needed to explore the notion of breakpoints and notification scheduling for tasks in domains that do not have specific goals structures, such as health care environments or safety critical domains.

9.7 Future Work

Future work includes:

Defining time windows for notification

One of the underlying assumptions of our proposed method of notification scheduling is that each notification request will have a time window associated with it, within which the notification will have to be delivered in order to be still valuable to the user. A second assumption is that the value of the notification remains the same throughout the duration of the time window, meaning that regardless of when within the time window the notification is delivered, the benefit to the user will be the same. Although this assumption may have been an over-simplification of how the value of a notification may change across time in real life settings, we believed that it to be a safe assumption, based on ongoing work on specifying urgency of notifications which does not change over time

(Horvitz, Jacobs et al. 1999). Determination of the time window was not a focus of this dissertation. However, for realization of a system like ours in practical settings, this is an important issue to resolve and presents new research challenges. For example, senders of notifications may manually enter the endpoint of the time window, or this information may be automatically gleaned from the content of the notification using semantic analysis. Existing work that looks at defining the urgency of emails uses natural language processing techniques to glean information from the email content, such as sender, recipient, time criticality etc (Horvitz, Jacobs et al. 1999). A similar approach coupled with sender intervention could be used to specify time windows for notifications.

Even if time windows can be defined with reasonable ease, it is not entirely clear that the value of the notification will be sustained throughout the entire time window. For example, the value of the notification may start diminishing towards the end of the time window, or may change during the time window based on the ongoing activity of the user. These uncertainties present opportunities for future research in this direction, and results will be valuable for improving performance of notification scheduling systems.

Improve various defer-to-breakpoint policies.

Our evaluation of notification scheduling using an automated framework only explored the basic defer-to-breakpoint policies, namely, interruption at Coarse/Medium/Fine breakpoints. However, in practical settings where the urgency of the notification will also be needed to be considered in the decision making of the system, more complex defer-to-breakpoint policies need to be realized, as described in Chapter 8. Our system currently provides support for applying these policies. The next step is to perform a more thorough evaluation focusing on the interaction between the policies, and the urgency and relevance of notifications. For example, a potentially useful policy is deferring a notification to the best breakpoint within a given time window. This presents several interesting challenges. First, this requires the system to be able to forecast whether a better breakpoint will occur within the remaining time. To forecast, the system requires information of prior patterns of breakpoint distribution. There can be potentially many determinants of the patterns, e.g., application being used, time spent on application, type of task being performed, time of day, day of week, calendar schedules etc. Computing

probability distributions of breakpoint distances is a interesting problem in itself and in depth evaluation is needed to understand how well the predictions based on the distributions work.

Explore methods for presenting notifications

Even if notifications are timed so that the cost and benefits are balanced, there is a need to provide appropriate visualizations for notifications to facilitate better management of the information being conveyed. One possible solution approach could be through developing visual summaries of notifications, so that the appropriate balance between saliency and disruptiveness is achieved. Existing work explores visualizations that organize multiple notifications based on urgency. However, as notifications become more prevalent, visual summaries will need to be more 'intelligent', e.g., be more aware of the context in which they are presented. Some of the issues that we plan to investigate are determining what visual parameters of notifications impact perception, how to match utility and importance of the notification to an appropriate visualization, modality and location of the presentation and how to balance between saliency and disruption.

Explore how users react to intelligence

To be able to benefit from and trust an intelligent system making decisions on the user's behalf, it is important that users have a mental model of the mechanisms of the system (Tullio, Dey et al. 2007). As part of evaluating effects of deployment of our system, we plan to evaluate how well users perceive the underlying operations of the system. In particular, we are interested in exploring if users change their task execution patterns in response to the system – e.g. to invoke a certain type of system behavior. Early evidence that this may happen was seen in our evaluation studies in this dissertation. For example, we found users to intentionally save the document to invoke a breakpoint – so that they could get a notification at that point. Being able to understand the underlying mechanism of an intelligent system is important so that the user does not feel out of control at any given moment.

Chapter 10

Conclusion

The goal of this dissertation was to develop and evaluate computational techniques to better manage interruption in the desktop. Our work has made the following contributions in this domain. First, we have leveraged theories of attention to establish breakpoints during tasks as lower cost moments for interruption. Our work provides empirical evidence supporting prior theoretical postulations that breakpoints are opportune for interruption as they exhibit moments of lower workload during task execution. We also established that there are three types of such breakpoints at various levels of granularity of user tasks. This finding has important implications for interruption reasoning systems, particularly when the urgency and/or relevance of the notification need to be factored in scheduling decisions. For example, more urgent notifications can be delivered at a *fine* breakpoint which occur more frequently but have with slight higher cost where the less urgent ones may be deferred until the lower cost but less frequent *coarse* breakpoints. By incorporating defer-to-breakpoint policies interruption reasoning systems can leverage this flexibility in balancing benefits of notification and costs of interruption.

Second, we showed how the structure of a task can be leveraged to predict interruption costs at breakpoints. We have demonstrated that characteristics of the task structure, such as level of a breakpoint, difficulty of the preceding subtask and carry over from the subtasks surrounding the breakpoint can predict three levels of interruption cost with 63% accuracy. Our results also show that there is significant empirical benefit of interrupting at lower cost breakpoints as predicted by this model as compared to higher cost breakpoints. For tasks that have fixed execution sequences, this model provides a simple way of predicting cost of interrupting at each breakpoint a priori with *only* information about the task structure. Interruption reasoning systems can leverage such models as an inexpensive yet effective way of predicting interruption costs while deciding when to schedule notifications.

Third, we have developed a new technique to detect and differentiate three types of breakpoints during free-form interactive tasks without requiring any knowledge of the underlying task. The key innovation in this work is that it applies theories of event perception from cognitive science literature to identify breakpoints during interactive desktop tasks. As researchers had found with physical tasks, we found certain events to be commensurate with breakpoints (e.g. switching to an unrelated task or finishing editing code), making them more salient than other moments during task execution. An important finding was that without any expensive machinery, simply leveraging system and application level events within the desktop environment, three levels of breakpoints, Coarse, Medium and Fine, can be detected with high accuracy. For most desktop tasks that are free-form in nature, interruption reasoning systems can identify and differentiate different types of breakpoints using this technique.

Fourth, we have designed and developed OASIS, a framework for intelligently managing notifications. OASIS implements previously developed techniques for detecting and differentiating breakpoints without knowledge of the underlying task, and schedules notifications to occur at appropriate breakpoints based on notification deferral policies. The novelty of the system is that it embodies principles of mediating notification delivery based on cognitive structure of tasks and leverages moments (breakpoints) that are known to have lower processing load. Oasis allows techniques for interruption management that have been shown to work in the lab to be realized within authentic task settings, and provides a test bed for exploring various defer-to-breakpoint policies in context of the complexities arising from managing notifications in real environments.

Finally, we provided results and lessons from evaluating performance and effects of using our framework for scheduling notifications in authentic task settings. Our study provided positive effects of scheduling notifications as users performed complex tasks in two interactive domains, programming and diagram editing. Results showed reasonable performance of the system in detecting breakpoints during novel tasks performed by new users, but also highlighted the need to develop better models to improve accuracy in model differentiation. In a second study evaluating effects of scheduling notifications at breakpoints using our system, results showed that notifications at breakpoints had lower interruption costs in terms of frustration and reaction time compared to notifications

delivered randomly. We also found that the relevance of the notification content to the ongoing task determines the type of breakpoint at which it should be delivered, further highlighting the need to be able to detect and differentiate multiple levels of breakpoints. The core concept of scheduling notifications at breakpoints resonated with how users prefer notifications to be managed. This indicates that users would likely adopt the use of notification management systems such as Oasis in practice.

Through the design, development and evaluation of Oasis, our work provides the first evidence of how the mediated approach of notification management impacts interruption costs in authentic settings. Our results show encouraging positive impact of scheduling notifications at breakpoints, and opens up many new avenues for future research in this domain.

Appendix A
List of Candidate Features Used for Developing Statistical Models

List of candidate features for the General model:

```
sessionStartEvent-1secBefore
sessionStopEvent-1secBefore
querySessionUserLogOffEvent-1secBefore
querySessionSysShutDownEvent-1secBefore
windowSessionControlsEvent-1secBefore
toastCreateEvent-1secBefore
toastDestroyEvent-1secBefore
windowMovedEvent-1secBefore
windowMinimizedEvent-1secBefore
switchToMailEvent-1secBefore
switchToSearchEngineEvent-1secBefore
switchToEntertainmentAppEvent-1secBefore
switchToInternetShoppingEvent-1secBefore
switchToIMCLient-1secBefore
switchToWindowExplorer-1secBefore
switchToOtherAppEvent-1secBefore
lowInactivityRate-1secBefore
mediumInactivityRate-1secBefore
highInactivityRate-1secBefore
sessionStartEvent-2secBefore
sessionStopEvent-2secBefore
querySessionUserLogOffEvent-2secBefore
querySessionSysShutDownEvent-2secBefore
windowSessionControlsEvent-2secBefore
toastCreateEvent-2secBefore
toastDestroyEvent-2secBefore
windowMovedEvent-2secBefore
windowMinimizedEvent-2secBefore
switchToMailEvent-2secBefore
switchToSearchEngineEvent-2secBefore
switchToEntertainmentAppEvent-2secBefore
switchToInternetShoppingEvent-2secBefore
switchToIMCLient-2secBefore
switchToWindowExplorer-2secBefore
switchToOtherAppEvent-2secBefore
lowInactivityRate-2secBefore
mediumInactivityRate-2secBefore
highInactivityRate-2secBefore
sessionStartEvent-5secBefore
sessionStopEvent-5secBefore
querySessionUserLogOffEvent-5secBefore
```

querySessionSysShutDownEvent-5secBefore
windowSessionControlsEvent-5secBefore
toastCreateEvent-5secBefore
toastDestroyEvent-5secBefore
windowMovedEvent-5secBefore
windowMinimizedEvent-5secBefore
switchToMailEvent-5secBefore
switchToSearchEngineEvent-5secBefore
switchToEntertainmentAppEvent-5secBefore
switchToInternetShoppingEvent-5secBefore
switchToIMCLient-5secBefore
switchToWindowExplorer-5secBefore
switchToOtherAppEvent-5secBefore
lowInactivityRate-5secBefore
mediumInactivityRate-5secBefore
highInactivityRate-5secBefore
sessionStartEvent-10secBefore
sessionStopEvent-10secBefore
querySessionUserLogOffEvent-10secBefore
querySessionSysShutDownEvent-10secBefore
windowSessionControlsEvent-10secBefore
toastCreateEvent-10secBefore
toastDestroyEvent-10secBefore
windowMovedEvent-10secBefore
windowMinimizedEvent-10secBefore
switchToMailEvent-10secBefore
switchToSearchEngineEvent-10secBefore
switchToEntertainmentAppEvent-10secBefore
switchToInternetShoppingEvent-10secBefore
switchToIMCLient-10secBefore
switchToWindowExplorer-10secBefore
switchToOtherAppEvent-10secBefore
lowInactivityRate-10secBefore
mediumInactivityRate-10secBefore
highInactivityRate-10secBefore
userInactiveDuringBreakpoint
relatedSwitches
unrelatedSwitches

List of candidate features for the Diagram Editing model for Medium and Fine:

eBeforeSuspend-1secBefore
eVisioIdle-1secBefore
eAppActivated-1secBefore
eAppDeactivated-1secBefore
eBeforeQuit-1secBefore
eAfterModal-1secBefore
eWindowActivated-1secBefore
eWindowOpened-1secBefore
eWindowChanged-1secBefore
eBeforeWindowClosed-1secBefore
eBeforeDocumentClose-1secBefore

eDocumentCreated-1secBefore
eDocumentSaved-1secBefore
eDocumentSavedAs-1secBefore
eDocumentOpened-1secBefore
eDocumentChanged-1secBefore
eBeforeDataRecordSetDelete-1secBefore
eDataRecordSetChanged-1secBefore
eDataRecordSetAdded-1secBefore
eEnterScope-1secBefore
eExitScope-1secBefore
eBeforeMasterDelete-1secBefore
eBeforeMasterDeleteCancel-1secBefore
eMasterChanged-1secBefore
eMasterAdded-1secBefore
eBeforePageDelete-1secBefore
ePageDeleteCancelled-1secBefore
ePageChanged-1secBefore
ePageAdded-1secBefore
eGroupCancelled-1secBefore
eConvertToGroupCancelled-1secBefore
eConnectionsDeleted-1secBefore
eConnectionsAdded-1secBefore
eBeforeStyleDelete-1secBefore
eBeforeShapeTextEdit-1secBefore
eBeforeShapeDelete-1secBefore
eBeforeSelectionDelete-1secBefore
eViewChangeBegin-1secBefore
eViewChangeEnd-1secBefore
eUngroupCancelled-1secBefore
eTextChanged-1secBefore
eStyleDeleteCancelled-1secBefore
eStyleChanged-1secBefore
eStyleAdded-1secBefore
eShapeParentChanged-1secBefore
eShapeLinkDeleted-1secBefore
eShapeLinkAdded-1secBefore
eShapeExitedTextEdit-1secBefore
eShapeDataGraphicChanged-1secBefore
eShapeChanged-1secBefore
eShapeAdded-1secBefore
eSelectionAdded-1secBefore
eSelectionChanged-1secBefore
eMouseMoveBegin-1secBefore
eMouseMoveEnd-1secBefore
lowCutRate-1secBefore
moderateCutRate-1secBefore
highCutRate-1secBefore
lowCopyRate-1secBefore
moderateCopyRate-1secBefore
highCopyRate-1secBefore
lowPasteRate-1secBefore
moderatePasteRate-1secBefore
highPasteRate-1secBefore
lowMoveRate-1secBefore

moderateMoveRate-1secBefore
highMoveRate-1secBefore
lowSizeRate-1secBefore
moderateSizeRate-1secBefore
highSizeRate-1secBefore
lowTextRate-1secBefore
moderateTextRate-1secBefore
highTextRate-1secBefore
lowDropRate-1secBefore
moderateDropRate-1secBefore
highDropRate-1secBefore
lowMouseMoveRate-1secBefore
moderateMouseMoveRate-1secBefore
highMouseMoveRate-1secBefore
lowMouseDirectionChangeRate-1secBefore
moderateMouseDirectionChangeRate-1secBefore
highMouseDirectionChangeRate-1secBefore
lowMouseDistanceTravelledRate-1secBefore
moderateMouseDistanceTravelledRate-1secBefore
highMouseDistanceTravelledRate-1secBefore
lowMouseVelocityRate-1secBefore
moderateMouseVelocityRate-1secBefore
highMouseVelocityRate-1secBefore
eBeforeSuspend-2secBefore
eVisioIdle-2secBefore
eAppActivated-2secBefore
eAppDeactivated-2secBefore
eBeforeQuit-2secBefore
eAfterModal-2secBefore
eWindowActivated-2secBefore
eWindowOpened-2secBefore
eWindowChanged-2secBefore
eBeforeWindowClosed-2secBefore
eBeforeDocumentClose-2secBefore
eDocumentCreated-2secBefore
eDocumentSaved-2secBefore
eDocumentSavedAs-2secBefore
eDocumentOpened-2secBefore
eDocumentChanged-2secBefore
eBeforeDataRecordSetDelete-2secBefore
eDataRecordSetChanged-2secBefore
eDataRecordSetAdded-2secBefore
eEnterScope-2secBefore
eExitScope-2secBefore
eBeforeMasterDelete-2secBefore
eBeforeMasterDeleteCancel-2secBefore
eMasterChanged-2secBefore
eMasterAdded-2secBefore
eBeforePageDelete-2secBefore
ePageDeleteCancelled-2secBefore
ePageChanged-2secBefore
ePageAdded-2secBefore
eGroupCancelled-2secBefore
eConvertToGroupCancelled-2secBefore

eConnectionsDeleted-2secBefore
eConnectionsAdded-2secBefore
eBeforeStyleDelete-2secBefore
eBeforeShapeTextEdit-2secBefore
eBeforeShapeDelete-2secBefore
eBeforeSelectionDelete-2secBefore
eViewChangeBegin-2secBefore
eViewChangeEnd-2secBefore
eUngroupCancelled-2secBefore
eTextChanged-2secBefore
eStyleDeleteCancelled-2secBefore
eStyleChanged-2secBefore
eStyleAdded-2secBefore
eShapeParentChanged-2secBefore
eShapeLinkDeleted-2secBefore
eShapeLinkAdded-2secBefore
eShapeExitedTextEdit-2secBefore
eShapeDataGraphicChanged-2secBefore
eShapeChanged-2secBefore
eShapeAdded-2secBefore
eSelectionAdded-2secBefore
eSelectionChanged-2secBefore
eMouseMoveBegin-2secBefore
eMouseMoveEnd-2secBefore
lowCutRate-2secBefore
moderateCutRate-2secBefore
highCutRate-2secBefore
lowCopyRate-2secBefore
moderateCopyRate-2secBefore
highCopyRate-2secBefore
lowPasteRate-2secBefore
moderatePasteRate-2secBefore
highPasteRate-2secBefore
lowMoveRate-2secBefore
moderateMoveRate-2secBefore
highMoveRate-2secBefore
lowSizeRate-2secBefore
moderateSizeRate-2secBefore
highSizeRate-2secBefore
lowTextRate-2secBefore
moderateTextRate-2secBefore
highTextRate-2secBefore
lowDropRate-2secBefore
moderateDropRate-2secBefore
highDropRate-2secBefore
lowMouseMoveRate-2secBefore
moderateMouseMoveRate-2secBefore
highMouseMoveRate-2secBefore
lowMouseDirectionChangeRate-2secBefore
moderateMouseDirectionChangeRate-2secBefore
highMouseDirectionChangeRate-2secBefore
lowMouseDistanceTravelledRate-2secBefore
moderateMouseDistanceTravelledRate-2secBefore
highMouseDistanceTravelledRate-2secBefore

```
lowMouseVelocityRate-2secBefore
moderateMouseVelocityRate-2secBefore
highMouseVelocityRate-2secBefore
eBeforeSuspend-5secBefore
eVisioIdle-5secBefore
eAppActivated-5secBefore
eAppDeactivated-5secBefore
eBeforeQuit-5secBefore
eAfterModal-5secBefore
eWindowActivated-5secBefore
eWindowOpened-5secBefore
eWindowChanged-5secBefore
eBeforeWindowClosed-5secBefore
eBeforeDocumentClose-5secBefore
eDocumentCreated-5secBefore
eDocumentSaved-5secBefore
eDocumentSavedAs-5secBefore
eDocumentOpened-5secBefore
eDocumentChanged-5secBefore
eBeforeDataRecordSetDelete-5secBefore
eDataRecordSetChanged-5secBefore
eDataRecordSetAdded-5secBefore
eEnterScope-5secBefore
eExitScope-5secBefore
eBeforeMasterDelete-5secBefore
eBeforeMasterDeleteCancel-5secBefore
eMasterChanged-5secBefore
eMasterAdded-5secBefore
eBeforePageDelete-5secBefore
ePageDeleteCancelled-5secBefore
ePageChanged-5secBefore
ePageAdded-5secBefore
eGroupCancelled-5secBefore
eConvertToGroupCancelled-5secBefore
eConnectionsDeleted-5secBefore
eConnectionsAdded-5secBefore
eBeforeStyleDelete-5secBefore
eBeforeShapeTextEdit-5secBefore
eBeforeShapeDelete-5secBefore
eBeforeSelectionDelete-5secBefore
eViewChangeBegin-5secBefore
eViewChangeEnd-5secBefore
eUngroupCancelled-5secBefore
eTextChanged-5secBefore
eStyleDeleteCancelled-5secBefore
eStyleChanged-5secBefore
eStyleAdded-5secBefore
eShapeParentChanged-5secBefore
eShapeLinkDeleted-5secBefore
eShapeLinkAdded-5secBefore
eShapeExitedTextEdit-5secBefore
eShapeDataGraphicChanged-5secBefore
eShapeChanged-5secBefore
eShapeAdded-5secBefore
```

eSelectionAdded-5secBefore
eSelectionChanged-5secBefore
eMouseMoveBegin-5secBefore
eMouseMoveEnd-5secBefore
lowCutRate-5secBefore
moderateCutRate-5secBefore
highCutRate-5secBefore
lowCopyRate-5secBefore
moderateCopyRate-5secBefore
highCopyRate-5secBefore
lowPasteRate-5secBefore
moderatePasteRate-5secBefore
highPasteRate-5secBefore
lowMoveRate-5secBefore
moderateMoveRate-5secBefore
highMoveRate-5secBefore
lowSizeRate-5secBefore
moderateSizeRate-5secBefore
highSizeRate-5secBefore
lowTextRate-5secBefore
moderateTextRate-5secBefore
highTextRate-5secBefore
lowDropRate-5secBefore
moderateDropRate-5secBefore
highDropRate-5secBefore
lowMouseMoveRate-5secBefore
moderateMouseMoveRate-5secBefore
highMouseMoveRate-5secBefore
lowMouseDirectionChangeRate-5secBefore
moderateMouseDirectionChangeRate-5secBefore
highMouseDirectionChangeRate-5secBefore
lowMouseDistanceTravelledRate-5secBefore
moderateMouseDistanceTravelledRate-5secBefore
highMouseDistanceTravelledRate-5secBefore
lowMouseVelocityRate-5secBefore
moderateMouseVelocityRate-5secBefore
highMouseVelocityRate-5secBefore
eBeforeSuspend-10secBefore
eVisioIdle-10secBefore
eAppActivated-10secBefore
eAppDeactivated-10secBefore
eBeforeQuit-10secBefore
eAfterModal-10secBefore
eWindowActivated-10secBefore
eWindowOpened-10secBefore
eWindowChanged-10secBefore
eBeforeWindowClosed-10secBefore
eBeforeDocumentClose-10secBefore
eDocumentCreated-10secBefore
eDocumentSaved-10secBefore
eDocumentSavedAs-10secBefore
eDocumentOpened-10secBefore
eDocumentChanged-10secBefore
eBeforeDataRecordSetDelete-10secBefore

eDataRecordSetChanged-10secBefore
eDataRecordSetAdded-10secBefore
eEnterScope-10secBefore
eExitScope-10secBefore
eBeforeMasterDelete-10secBefore
eBeforeMasterDeleteCancel-10secBefore
eMasterChanged-10secBefore
eMasterAdded-10secBefore
eBeforePageDelete-10secBefore
ePageDeleteCancelled-10secBefore
ePageChanged-10secBefore
ePageAdded-10secBefore
eGroupCancelled-10secBefore
eConvertToGroupCancelled-10secBefore
eConnectionsDeleted-10secBefore
eConnectionsAdded-10secBefore
eBeforeStyleDelete-10secBefore
eBeforeShapeTextEdit-10secBefore
eBeforeShapeDelete-10secBefore
eBeforeSelectionDelete-10secBefore
eViewChangeBegin-10secBefore
eViewChangeEnd-10secBefore
eUngroupCancelled-10secBefore
eTextChanged-10secBefore
eStyleDeleteCancelled-10secBefore
eStyleChanged-10secBefore
eStyleAdded-10secBefore
eShapeParentChanged-10secBefore
eShapeLinkDeleted-10secBefore
eShapeLinkAdded-10secBefore
eShapeExitedTextEdit-10secBefore
eShapeDataGraphicChanged-10secBefore
eShapeChanged-10secBefore
eShapeAdded-10secBefore
eSelectionAdded-10secBefore
eSelectionChanged-10secBefore
eMouseMoveBegin-10secBefore
eMouseMoveEnd-10secBefore
lowCutRate-10secBefore
moderateCutRate-10secBefore
highCutRate-10secBefore
lowCopyRate-10secBefore
moderateCopyRate-10secBefore
highCopyRate-10secBefore
lowPasteRate-10secBefore
moderatePasteRate-10secBefore
highPasteRate-10secBefore
lowMoveRate-10secBefore
moderateMoveRate-10secBefore
highMoveRate-10secBefore
lowSizeRate-10secBefore
moderateSizeRate-10secBefore
highSizeRate-10secBefore
lowTextRate-10secBefore

moderateTextRate-10secBefore
highTextRate-10secBefore
lowDropRate-10secBefore
moderateDropRate-10secBefore
highDropRate-10secBefore
lowMouseMoveRate-10secBefore
moderateMouseMoveRate-10secBefore
highMouseMoveRate-10secBefore
lowMouseDirectionChangeRate-10secBefore
moderateMouseDirectionChangeRate-10secBefore
highMouseDirectionChangeRate-10secBefore
lowMouseDistanceTravelledRate-10secBefore
moderateMouseDistanceTravelledRate-10secBefore
highMouseDistanceTravelledRate-10secBefore
lowMouseVelocityRate-10secBefore
moderateMouseVelocityRate-10secBefore
highMouseVelocityRate-10secBefore
sessionStartEvent-1secBefore
sessionStopEvent-1secBefore
querySessionUserLogOffEvent-1secBefore
querySessionSysShutDownEvent-1secBefore
windowSessionControlsEvent-1secBefore
toastCreateEvent-1secBefore
toastDestroyEvent-1secBefore
windowMovedEvent-1secBefore
windowMinimizedEvent-1secBefore
switchToMailEvent-1secBefore
switchToSearchEngineEvent-1secBefore
switchToEntertainmentAppEvent-1secBefore
switchToInternetShoppingEvent-1secBefore
switchToIMCLient-1secBefore
switchToWindowExplorer-1secBefore
switchToOtherAppEvent-1secBefore
lowInactivityRate-1secBefore
mediumInactivityRate-1secBefore
highInactivityRate-1secBefore
sessionStartEvent-2secBefore
sessionStopEvent-2secBefore
querySessionUserLogOffEvent-2secBefore
querySessionSysShutDownEvent-2secBefore
windowSessionControlsEvent-2secBefore
toastCreateEvent-2secBefore
toastDestroyEvent-2secBefore
windowMovedEvent-2secBefore
windowMinimizedEvent-2secBefore
switchToMailEvent-2secBefore
switchToSearchEngineEvent-2secBefore
switchToEntertainmentAppEvent-2secBefore
switchToInternetShoppingEvent-2secBefore
switchToIMCLient-2secBefore
switchToWindowExplorer-2secBefore
switchToOtherAppEvent-2secBefore
lowInactivityRate-2secBefore
mediumInactivityRate-2secBefore

highInactivityRate-2secBefore
sessionStartEvent-5secBefore
sessionStopEvent-5secBefore
querySessionUserLogOffEvent-5secBefore
querySessionSysShutDownEvent-5secBefore
windowSessionControlsEvent-5secBefore
toastCreateEvent-5secBefore
toastDestroyEvent-5secBefore
windowMovedEvent-5secBefore
windowMinimizedEvent-5secBefore
switchToMailEvent-5secBefore
switchToSearchEngineEvent-5secBefore
switchToEntertainmentAppEvent-5secBefore
switchToInternetShoppingEvent-5secBefore
switchToIMCLient-5secBefore
switchToWindowExplorer-5secBefore
switchToOtherAppEvent-5secBefore
lowInactivityRate-5secBefore
mediumInactivityRate-5secBefore
highInactivityRate-5secBefore
sessionStartEvent-10secBefore
sessionStopEvent-10secBefore
querySessionUserLogOffEvent-10secBefore
querySessionSysShutDownEvent-10secBefore
windowSessionControlsEvent-10secBefore
toastCreateEvent-10secBefore
toastDestroyEvent-10secBefore
windowMovedEvent-10secBefore
windowMinimizedEvent-10secBefore
switchToMailEvent-10secBefore
switchToSearchEngineEvent-10secBefore
switchToEntertainmentAppEvent-10secBefore
switchToInternetShoppingEvent-10secBefore
switchToIMCLient-10secBefore
switchToWindowExplorer-10secBefore
switchToOtherAppEvent-10secBefore
lowInactivityRate-10secBefore
mediumInactivityRate-10secBefore
highInactivityRate-10secBefore
userInactiveDuringBreakpoint
relatedSwitches
unrelatedSwitches

List of candidate features for the Programming model for Medium and Fine:

cutActions-1secBefore
copyActions-1secBefore
pasteActions-1secBefore
navigationActions-1secBefore
otherKeyActions-1secBefore
ElementDeleted-1secBefore
ElementChanged-1secBefore
ElementAdded-1secBefore

docSaved-1secBefore
docOpened-1secBefore
docClosing-1secBefore
taskRemoved-1secBefore
taskNavigated-1secBefore
taskModified-1secBefore
taskAdded-1secBefore
windowCreated-1secBefore
windowClosing-1secBefore
windowActivated-1secBefore
projectConfigDone-1secBefore
buildDone-1secBefore
debugContextChanged-1secBefore
modeChanged-1secBefore
beginShutDown-1secBefore
macroRuntimeReset-1secBefore
startupComplete-1secBefore
findDone-1secBefore
selectionChanged-1secBefore
solutionRenamed-1secBefore
projectRenamedInSolution-1secBefore
projectRemovedFromSolution-1secBefore
projectAddedToSolution-1secBefore
solutionOpened-1secBefore
afterClosing-1secBefore
functionChanged-1secBefore
classChanged-1secBefore
modeChanged-1secBefore
mouseMoves-1secBefore
lowCompilationErrors-1secBefore
medCompilationErrors-1secBefore
highCompilationErrors-1secBefore
lowCutRate-1secBefore
moderateCutRate-1secBefore
highCutRate-1secBefore
lowCopyRate-1secBefore
moderateCopyRate-1secBefore
highCopyRate-1secBefore
lowPasteRate-1secBefore
moderatePasteRate-1secBefore
highPasteRate-1secBefore
lowNavRate-1secBefore
moderateNavRate-1secBefore
highNavRate-1secBefore
lowMouseMoveRate-1secBefore
moderateMouseMoveRate-1secBefore
highMouseMoveRate-1secBefore
lowMouseDirectionChangeRate-1secBefore
moderateMouseDirectionChangeRate-1secBefore
highMouseDirectionChangeRate-1secBefore
lowMouseDistanceTravelledRate-1secBefore
moderateMouseDistanceTravelledRate-1secBefore
highMouseDistanceTravelledRate-1secBefore
lowMouseVelocityRate-1secBefore

moderateMouseVelocityRate-1secBefore
highMouseVelocityRate-1secBefore
cutActions-2secBefore
copyActions-2secBefore
pasteActions-2secBefore
navigationActions-2secBefore
otherKeyActions-2secBefore
ElementDeleted-2secBefore
ElementChanged-2secBefore
ElementAdded-2secBefore
docSaved-2secBefore
docOpened-2secBefore
docClosing-2secBefore
taskRemoved-2secBefore
taskNavigated-2secBefore
taskModified-2secBefore
taskAdded-2secBefore
windowCreated-2secBefore
windowClosing-2secBefore
windowActivated-2secBefore
projectConfigDone-2secBefore
buildDone-2secBefore
debugContextChanged-2secBefore
modeChanged-2secBefore
beginShutDown-2secBefore
macroRuntimeReset-2secBefore
startupComplete-2secBefore
findDone-2secBefore
selectionChanged-2secBefore
solutionRenamed-2secBefore
projectRenamedInSolution-2secBefore
projectRemovedFromSolution-2secBefore
projectAddedToSolution-2secBefore
solutionOpened-2secBefore
afterClosing-2secBefore
functionChanged-2secBefore
classChanged-2secBefore
modeChanged-2secBefore
mouseMoves-2secBefore
lowCompilationErrors-2secBefore
medCompilationErrors-2secBefore
highCompilationErrors-2secBefore
lowCutRate-2secBefore
moderateCutRate-2secBefore
highCutRate-2secBefore
lowCopyRate-2secBefore
moderateCopyRate-2secBefore
highCopyRate-2secBefore
lowPasteRate-2secBefore
moderatePasteRate-2secBefore
highPasteRate-2secBefore
lowNavRate-2secBefore
moderateNavRate-2secBefore
highNavRate-2secBefore

lowMouseMoveRate-2secBefore
moderateMouseMoveRate-2secBefore
highMouseMoveRate-2secBefore
lowMouseDirectionChangeRate-2secBefore
moderateMouseDirectionChangeRate-2secBefore
highMouseDirectionChangeRate-2secBefore
lowMouseDistanceTravelledRate-2secBefore
moderateMouseDistanceTravelledRate-2secBefore
highMouseDistanceTravelledRate-2secBefore
lowMouseVelocityRate-2secBefore
moderateMouseVelocityRate-2secBefore
highMouseVelocityRate-2secBefore
cutActions-5secBefore
copyActions-5secBefore
pasteActions-5secBefore
navigationActions-5secBefore
otherKeyActions-5secBefore
ElementDeleted-5secBefore
ElementChanged-5secBefore
ElementAdded-5secBefore
docSaved-5secBefore
docOpened-5secBefore
docClosing-5secBefore
taskRemoved-5secBefore
taskNavigated-5secBefore
taskModified-5secBefore
taskAdded-5secBefore
windowCreated-5secBefore
windowClosing-5secBefore
windowActivated-5secBefore
projectConfigDone-5secBefore
buildDone-5secBefore
debugContextChanged-5secBefore
modeChanged-5secBefore
beginShutDown-5secBefore
macroRuntimeReset-5secBefore
startupComplete-5secBefore
findDone-5secBefore
selectionChanged-5secBefore
solutionRenamed-5secBefore
projectRenamedInSolution-5secBefore
projectRemovedFromSolution-5secBefore
projectAddedToSolution-5secBefore
solutionOpened-5secBefore
afterClosing-5secBefore
functionChanged-5secBefore
classChanged-5secBefore
modeChanged-5secBefore
mouseMoves-5secBefore
lowCompilationErrors-5secBefore
medCompilationErrors-5secBefore
highCompilationErrors-5secBefore
lowCutRate-5secBefore
moderateCutRate-5secBefore

highCutRate-5secBefore
lowCopyRate-5secBefore
moderateCopyRate-5secBefore
highCopyRate-5secBefore
lowPasteRate-5secBefore
moderatePasteRate-5secBefore
highPasteRate-5secBefore
lowNavRate-5secBefore
moderateNavRate-5secBefore
highNavRate-5secBefore
lowMouseMoveRate-5secBefore
moderateMouseMoveRate-5secBefore
highMouseMoveRate-5secBefore
lowMouseDirectionChangeRate-5secBefore
moderateMouseDirectionChangeRate-5secBefore
highMouseDirectionChangeRate-5secBefore
lowMouseDistanceTravelledRate-5secBefore
moderateMouseDistanceTravelledRate-5secBefore
highMouseDistanceTravelledRate-5secBefore
lowMouseVelocityRate-5secBefore
moderateMouseVelocityRate-5secBefore
highMouseVelocityRate-5secBefore
cutActions-10secBefore
copyActions-10secBefore
pasteActions-10secBefore
navigationActions-10secBefore
otherKeyActions-10secBefore
ElementDeleted-10secBefore
ElementChanged-10secBefore
ElementAdded-10secBefore
docSaved-10secBefore
docOpened-10secBefore
docClosing-10secBefore
taskRemoved-10secBefore
taskNavigated-10secBefore
taskModified-10secBefore
taskAdded-10secBefore
windowCreated-10secBefore
windowClosing-10secBefore
windowActivated-10secBefore
projectConfigDone-10secBefore
buildDone-10secBefore
debugContextChanged-10secBefore
modeChanged-10secBefore
beginShutDown-10secBefore
macroRuntimeReset-10secBefore
startupComplete-10secBefore
findDone-10secBefore
selectionChanged-10secBefore
solutionRenamed-10secBefore
projectRenamedInSolution-10secBefore
projectRemovedFromSolution-10secBefore
projectAddedToSolution-10secBefore
solutionOpened-10secBefore

afterClosing-10secBefore
functionChanged-10secBefore
classChanged-10secBefore
modeChanged-10secBefore
mouseMoves-10secBefore
lowCompilationErrors-10secBefore
medCompilationErrors-10secBefore
highCompilationErrors-10secBefore
lowCutRate-10secBefore
moderateCutRate-10secBefore
highCutRate-10secBefore
lowCopyRate-10secBefore
moderateCopyRate-10secBefore
highCopyRate-10secBefore
lowPasteRate-10secBefore
moderatePasteRate-10secBefore
highPasteRate-10secBefore
lowNavRate-10secBefore
moderateNavRate-10secBefore
highNavRate-10secBefore
lowMouseMoveRate-10secBefore
moderateMouseMoveRate-10secBefore
highMouseMoveRate-10secBefore
lowMouseDirectionChangeRate-10secBefore
moderateMouseDirectionChangeRate-10secBefore
highMouseDirectionChangeRate-10secBefore
lowMouseDistanceTravelledRate-10secBefore
moderateMouseDistanceTravelledRate-10secBefore
highMouseDistanceTravelledRate-10secBefore
lowMouseVelocityRate-10secBefore
moderateMouseVelocityRate-10secBefore
highMouseVelocityRate-10secBefore
sessionStartEvent-1secBefore
sessionStopEvent-1secBefore
querySessionUserLogOffEvent-1secBefore
querySessionSysShutDownEvent-1secBefore
windowSessionControlsEvent-1secBefore
toastCreateEvent-1secBefore
toastDestroyEvent-1secBefore
windowMovedEvent-1secBefore
windowMinimizedEvent-1secBefore
switchToMailEvent-1secBefore
switchToSearchEngineEvent-1secBefore
switchToEntertainmentAppEvent-1secBefore
switchToInternetShoppingEvent-1secBefore
switchToIMCLient-1secBefore
switchToWindowExplorer-1secBefore
switchToOtherAppEvent-1secBefore
lowInactivityRate-1secBefore
mediumInactivityRate-1secBefore
highInactivityRate-1secBefore
sessionStartEvent-2secBefore
sessionStopEvent-2secBefore
querySessionUserLogOffEvent-2secBefore

querySessionSysShutDownEvent-2secBefore
windowSessionControlsEvent-2secBefore
toastCreateEvent-2secBefore
toastDestroyEvent-2secBefore
windowMovedEvent-2secBefore
windowMinimizedEvent-2secBefore
switchToMailEvent-2secBefore
switchToSearchEngineEvent-2secBefore
switchToEntertainmentAppEvent-2secBefore
switchToInternetShoppingEvent-2secBefore
switchToIMCLient-2secBefore
switchToWindowExplorer-2secBefore
switchToOtherAppEvent-2secBefore
lowInactivityRate-2secBefore
mediumInactivityRate-2secBefore
highInactivityRate-2secBefore
sessionStartEvent-5secBefore
sessionStopEvent-5secBefore
querySessionUserLogOffEvent-5secBefore
querySessionSysShutDownEvent-5secBefore
windowSessionControlsEvent-5secBefore
toastCreateEvent-5secBefore
toastDestroyEvent-5secBefore
windowMovedEvent-5secBefore
windowMinimizedEvent-5secBefore
switchToMailEvent-5secBefore
switchToSearchEngineEvent-5secBefore
switchToEntertainmentAppEvent-5secBefore
switchToInternetShoppingEvent-5secBefore
switchToIMCLient-5secBefore
switchToWindowExplorer-5secBefore
switchToOtherAppEvent-5secBefore
lowInactivityRate-5secBefore
mediumInactivityRate-5secBefore
highInactivityRate-5secBefore
sessionStartEvent-10secBefore
sessionStopEvent-10secBefore
querySessionUserLogOffEvent-10secBefore
querySessionSysShutDownEvent-10secBefore
windowSessionControlsEvent-10secBefore
toastCreateEvent-10secBefore
toastDestroyEvent-10secBefore
windowMovedEvent-10secBefore
windowMinimizedEvent-10secBefore
switchToMailEvent-10secBefore
switchToSearchEngineEvent-10secBefore
switchToEntertainmentAppEvent-10secBefore
switchToInternetShoppingEvent-10secBefore
switchToIMCLient-10secBefore
switchToWindowExplorer-10secBefore
switchToOtherAppEvent-10secBefore
lowInactivityRate-10secBefore
mediumInactivityRate-10secBefore
highInactivityRate-10secBefore

```
userInactiveDuringBreakpoint
relatedSwitches
unrelatedSwitches
```

APPENDIX B
Artifacts Used in Studies Conducted in This Thesis

In this chapter we present artifacts used in the various studies conducted as part of this dissertation. We include recruitment scripts, sample task descriptions, questionnaires and sample output in chronological order.

Recruitment Script for Pupil Dilation study (Chapter 4):

Hello, all,

As you may know, we are working on the pupil dilation and cognitive load project. Now we are looking for participants for the second phase of our experiment.
The experiment will be run at the Beckman Institute, and it will take about 30 minutes at the most. We will use a helmet based eye tracker, the Eyelink II (http://www.eyelinkinfo.com/). You will have to perform two simple computer based tasks wearing the eye tracker. The eyetracker will measure your pupil area while you perform the tasks.

Due to the budget limits, we may not be able to pay you. But we will provide some homemade cookies to show our appreciation for your help. Possible schedules are,

Aug 11	Monday	7:00pm - 7:30 pm	7:30pm-8:00pm	8:00pm -8:30pm
Aug 12	Tuesday	7:00pm - 7:30 pm	7:30pm-8:00pm	8:00pm -8:30pm
Aug 13	Wednesday	7:00pm - 7:30 pm	7:30pm-8:00pm	8:00pm -8:30pm
Aug 14	Thursday	7:00pm - 7:30 pm	7:30pm-8:00pm	8:00pm -8:30pm

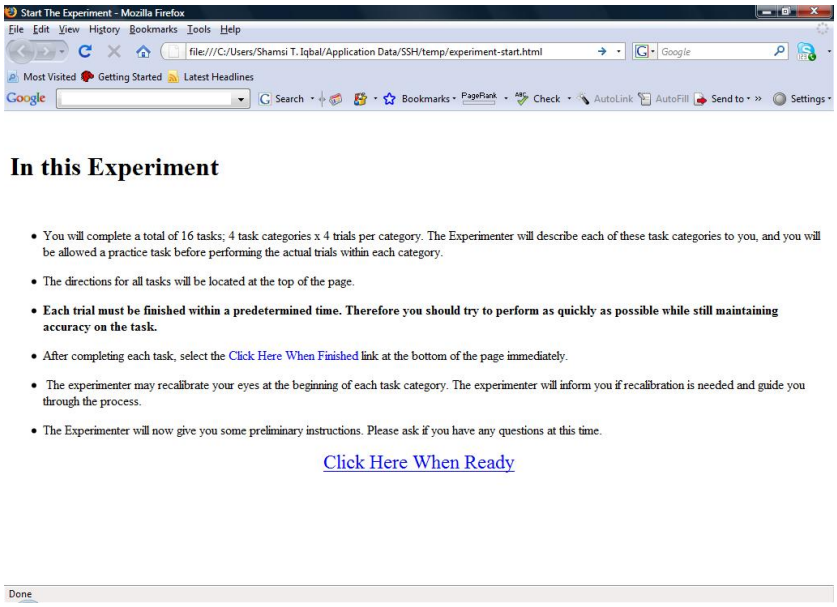
Venue:
1436 Beckman Institute (Cognitive Development Lab)
Tel: 244-7336 (Please ask for Sam)

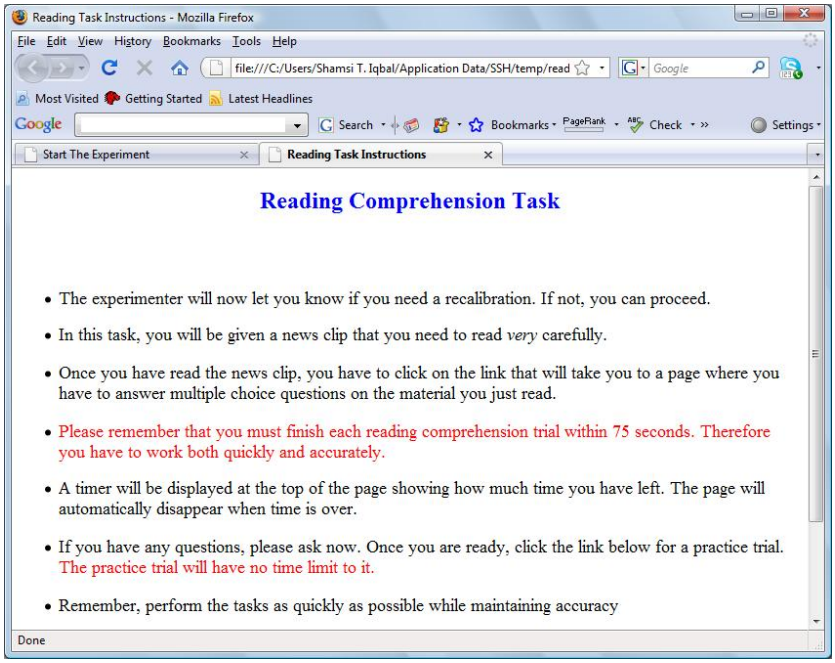
Please let us know if you are willing to help and what time works for you. We are flexible with time schedule as long as it is between 7pm and 8:30 pm.

Thanks in advance,

Shamsi (siqbal@uiuc.edu) and Sam(xzheng@uiuc.edu)

Sample Instructions for Pupil Dilation Study (Chapter 4):





Instructions for investigating effectiveness of pupil dilation as a predictor of mental workload (Chapter 4):

In this study we are going to measure how primary tasks are impacted through the occurrence of interruptions and whether the timing of the interruptions affects the primary task.

You will perform a series of 12 tasks from 3 task categories – route planning, document editing and object manipulation. The task categories will be presented to you randomly. Each task category will have instructions, a practice trial which will help you familiarize yourself with the task and 4 actual trials. You are encouraged to ask questions during the practice trial, but not during any of the actual trials.

During performing tasks, you may be interrupted with another task. You have to attend to the interrupting task immediately. The interrupting task will require you to read a news item and select the most suitable title for it from a list of three titles. Once you have completed the interrupting task, you will resume the task you were performing originally.

Please complete the tasks as quickly as possible while maintaining maximum accuracy.

At the end of each trial, you will fill out a questionnaire on your experience with the task just completed.

For analysis purposes, we will video tape this session and also record your onscreen activities. These videos will be used to find timestamps and performance details of user interactions. Please be assured that the videos will not be made public and will be viewed only by the experimenters. Your face will not be recorded in the video for anonymity purposes. We will destroy the video after completion of the analysis procedures.

Thank you and feel free to ask questions at this point.

TLX form to measure workload for investigating effectiveness of pupil dilation as a predictor of mental workload (Chapter 4):

To be filled out by the experimenter: **User __** **Task_____** **Interruption Timing ____** **Trial _____**

Task Questionnaire

Please indicate your ratings by marking a vertical line along each dimension. Your responses should be based on how you evaluate your experience with the task**.**

Mental Demand (thinking, deciding, remembering)

Low ——————————————————————————— High

Temporal Demand (how much time pressure you felt)

Low ——————————————————————————— High

Effort (how hard you worked for the task)

Low ——————————————————————————— High

Frustration (how frustrating was completing the task)

Low ——————————————————————————— High

Annoyance (how annoying was your experience with the task)

Low ——————————————————————————— High

Own Performance (success in accomplishing the task)

Poor ——————————————————————————— Good

Respect (how respectful the system was when presenting the **news** task)

Low ——————————————————————————— High

Instructions for study for developing task-independent statistical models to detect and differentiate breakpoints (Chapter 5)

In this study we are going to measure what factors of interruptions disrupt the primary task and how.

You will perform a series of 21 tasks from 3 task categories – route planning, document editing and video editing. The task categories will be presented to you randomly. Each task category will have instructions, a practice trial which will help you familiarize yourself with the task and a set of 6 actual trials. You are encouraged to ask questions only during the practice trial.

After each practice trial, you will be presented with the actual trial set. During performing tasks, you may be interrupted with another task. You have to attend to the interrupting task immediately. At the beginning of the interrupting task you have to rate the annoyance caused by and the disrespect shown by the interrupting task using a sliding scale on the monitor. Then you will proceed to the actual interrupting task. The task will require you to read a stock quote and make a decision based on the quote. Once you have completed the interrupting task, you will resume the task you were performing originally.

After completion of each trial you have to fill out an online questionnaire by clicking on a link on the bottom right corner of the page. You will click on the Next link to go to the next trial.

Please complete the tasks as quickly as possible while maintaining maximum accuracy.

For analysis purposes, we will video tape this session and also record your onscreen activities. These videos will be used to find timestamps and performance details of user interactions. Please be assured that the videos will not be made public and will be viewed only by the experimenters. We will destroy the video after completion of the analysis procedures.

Thank you and feel free to ask questions at this point.

Pre-experiment Questionnaire for study for leveraging characteristics of task structure to predict cost of interruption (Chapter 5)

User #:

Name:

Gender: M F

Age:

Occupation:

Major:

Questions:

1. Have you used Microsoft Word before? Y N

2. If your answer to 1 is yes, then please indicate your level of expertise with it:

Use very rarely Use Occasionally Regular user

3. Have you used Microsoft Excel before? Y N

4. If your answer to 3 is yes, then please indicate your level of expertise with it:

Use very rarely Use Occasionally Regular user

5. Have you used Windows Movie Maker before? Y N

6. If your answer to 5 is yes, then please indicate your level of expertise with it:

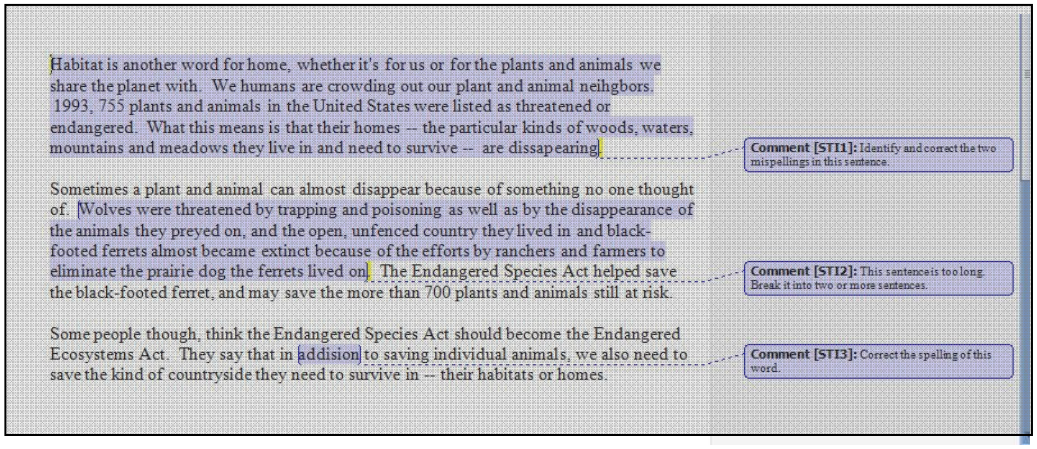
Use very rarely Use Occasionally Regular user

Sample Task Description for study for study for leveraging characteristics of task structure to predict cost of interruption (Chapter 5):

Document Editing Task - 2

- Click on the link below to open the word document for the task.
- When you have completed the task inform the experimenter.
- **Please perform the task as quickly as possible while maintaining accuracy on the task.**

Click here to begin

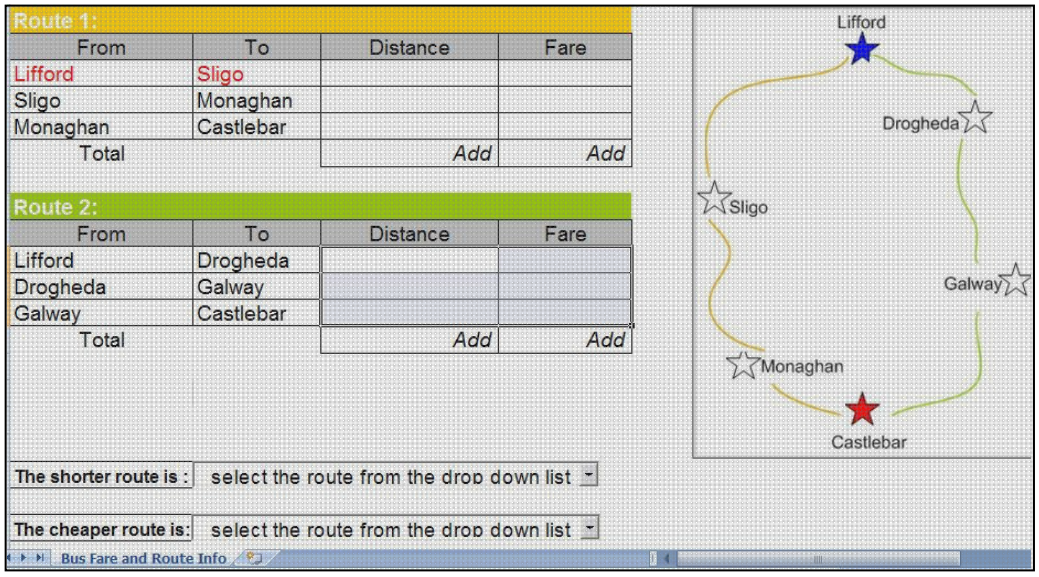


Sample Task Description for study for study for leveraging characteristics of task structure to predict cost of interruption (Chapter 5):

Route Planning Task - 3

- Click on the link below to open the excel document for the task.
- When you have completed the task inform the experimenter.
- **Please perform the task as quickly as possible while maintaining accuracy on the task.**

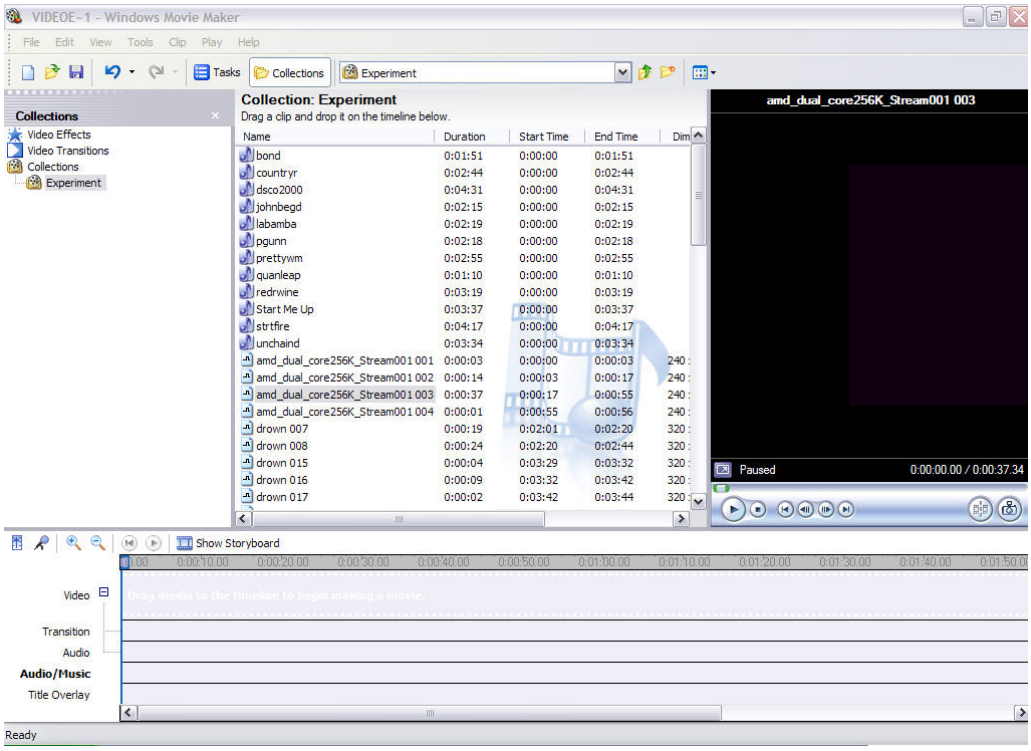
Click here to begin



Sample Task Description for study for study for leveraging characteristics of task structure to predict cost of interruption (Chapter 5):

Video Editing Task - 5

- For this task, you will create a video on *bike stunts*. The video and audio clips are already provided in the video project.
- Switch to Windows Movie Maker and start a new project.
- Click on **Bike Stunts** folder under the **Collections** folder to get the video and audio clips for this task..
- When you have completed the task **minimize** Windows Movie Maker and return to this screen.
- **Please perform the task as quickly as possible while maintaining accuracy on the task.**



Post-experiment Questionnaire for study for study for leveraging characteristics of task structure to predict cost of interruption (Chapter 5):

1. How did you feel about the interruptions during your primary tasks?

2. Did you notice a difference in when the interruption was presented across different tasks?

3. Did you feel that the interruptions could have been timed better?

4. Assume that the interruptions are important to you. If it was up to you, when would you have scheduled the interruptions to occur?

Instructions for validation study for study for leveraging characteristics of task structure to predict cost of interruption (Chapter 5):

In this study we are going to evaluate how well a cost of interruption model can predict disruptive effects of interruption.

You will perform a series of 8 tasks from 2 task categories – collage generation and form design. The task categories will be presented to you randomly. Each task category will have instructions, a practice trial which will help you familiarize yourself with the task and a set of 3 actual trials. You are encouraged to ask questions only during the practice trial.

After each practice trial, you will be presented with the actual trial set. During performing tasks, you may be interrupted with another task. You have to attend to the interrupting task immediately. At the beginning of the interrupting task you have to rate the annoyance caused by and the disrespect shown by the interrupting task using a sliding scale on the monitor. Then you will proceed to the actual interrupting task. The task will require you to read a stock quote and make a decision based on the quote. Once you have completed the interrupting task, you will resume the task you were performing originally.

Please complete the tasks as quickly as possible while being as creative as possible.

For analysis purposes, we will record your onscreen activities. These videos will be used to find timestamps and performance details of user interactions. Please be assured that the videos will not be made public and will be viewed only by the experimenters.

Thank you and feel free to ask questions at this point.

Sample Task Description for validation study for leveraging characteristics of task structure to predict cost of interruption (Chapter 5):

Form Design Task - 1

Customer Satisfaction Survey Form:

- You are required to design a customer satisfaction form for a popular restaurant. We have created part of the form and you have to complete it according to instructions. You should design the form in such a way so that the form elicits the necessary information without requiring the user to type too much. For example, use check boxes, list boxes, radio buttons to provide the user with choices rather than have the user type.

Instructions:

1. Open Adobe Designer. You will find a shortcut on the desktop.

2. Go to File-> Open and browse to Desktop. Select Adobe Design Template (*.tds) from the File Type drop down list. Open Customer.tds.

3. Add the following to the form:

   - Add appropriate questions evaluating the user's satisfaction with **Food.**
   - Do the same for **Service** and **Cost.**
   - Add the restaurant logo at the top left corner. You will find the logo on the desktop.

4. To evaluate your form, you can preview the form by clicking on the **PDF Preview** tab.

5. When you are finished, save the document on the desktop as Customer-final.tds.

Sample Task Description for validation study for leveraging characteristics of task structure to predict cost of interruption (Chapter 5):

Collage Generation Task - 2

CS Collage:

You are required to create a collage of images of the Computer Science department. This collage may be used, for example, on the departmental website for showcasing various aspects of the department.

Instructions:

1. Open Adobe Photoshop. You will find the shortcut on the desktop.

2. Go to File->New and open an empty document for the collage. The width and height of the document should be 450 pixels and 300 pixels respectively.

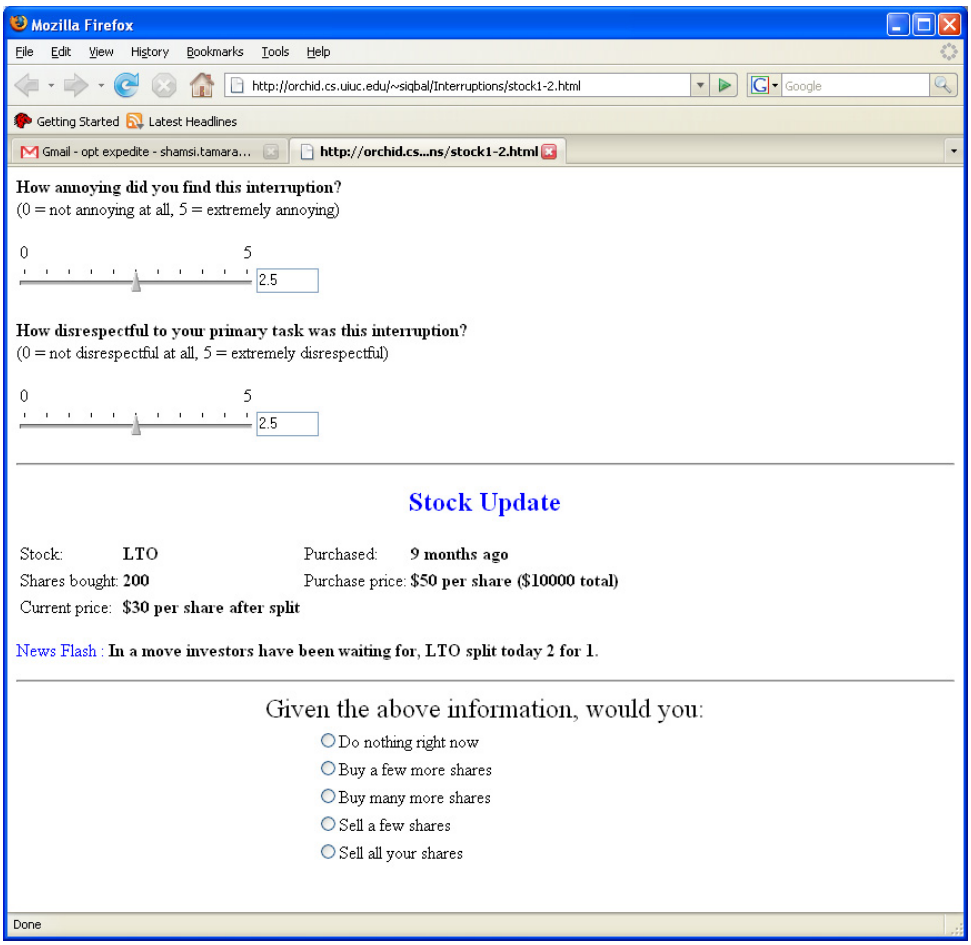
3. Go to File->Open->**CS** where all images for this collage are stored. For your convenience, please open all images from all four folders.

4. Browse through the open images and copy and paste the images you like into the collage. You must have at least one image from **Exterior,** one from **Interior,** one from **Research Labs** and one from **CS Life.**

5. Once you are done, add any effect of your choice on the collage. For example, you can soften the sharp edges by using the *blur* tool.

6. When you are finished, save the document as a jpeg file with any name of your choice in the **CS** directory.

Sample Interruption for validation study for leveraging characteristics of task structure to predict cost of interruption (Chapter 5):



Instructions for study for collecting data for developing task-independent statistical models to detect and differentiate breakpoints (Chapter 6):

Analysis of User Tasks in the Desktop Environment

In this study we will collect data on task execution patterns of desktop users. The data will be used to build models of interruptibility during task execution and will be later used to drive a system that reasons about when to interrupt.

For the study, you are requested to participate in two phases. Overview of the phases are described below:

Phase 1:

In the first phase, you will perform your own tasks (either of document editing/programming/image manipulation) in your own environment/machine where we will install screen and event capture software to record your onscreen activities. The recording can be at any time of your choice since you will start and stop the software, but need not exceed two hours. You can stop or pause recording using an easily accessible interface at any point during recording.

Since we want to capture natural behavior of users as much as possible, we encourage you to perform your tasks as you do regularly. For example, if you switch frequently between the email client and the document editor while writing a paper, you should do the same during the recording phase. However, we prefer that you choose a time for recording your activities when you know that your main focus will be on either one of the aforementioned primary tasks so that we get data pertaining to predominantly one task.

The collected data will be stored in two files. One is a video file which will be saved in **C:\Activity Recorder** as *"<your name-date-time>*.avi". The other one is a log file with all the keyboard, mouse and window events, saved as *HookData.txt* in **C:\ .**

You will copy these two files in the cd given to you by the experimenter and return it to the experimenter the same day if possible. This concludes phase 1.

Phase 2:

Within the same day or the next day at the latest you will come to 3113 Siebel Center for the second phase. In this phase, you will go through the video that you turned in previously and annotate different phases in the video. More specifically, we will ask you to identify moments in the video where you felt that you had mental pauses or breakpoints, using a custom built video annotation tool.

Your annotation data will be stored as separate log files in the experiment machine. Later, we will have independent observers look at your videos and identify breakpoints so that we can compare how much people can perceive about breakpoints without having experienced the task execution themselves.

Please be assured that your video files will be completely anonymized. If needed, during the annotation phase, you can point out parts of the video which you do not wish to be seen and we will edit those out of the video.

Constraints:

1. Windows XP user
2. Should have at least 1 GB of free space on their machine and a CD-R drive
3. Alternatively, can bring their work (programming, document editing, image manipulation) to 3113 and perform their tasks on the lab machine.
4. Document editing task must use Microsoft Word
5. Programming task must use Eclipse (preferable) or Visual Studio
6. Image Manipulation task must use Adobe Photoshop or GIMP

Sign up sheet: (sign up for only one slot)

Document Editing			**Programming**			**Image Manipulation**		
Name	Phase 1 date	Phase 2 date	Name	Phase 1 date	Phase 2 date	Name	Phase 1 date	Phase 2 date

Instructions for managing the data collection software (Chapter 6):

To start the software:

1. Double click the application installHook (should be in C:/Program Files/ ActivityDataCollector or something along those lines).
2. On the dialogue box that shows up, click on Start to start recording.
3. This will start two applications simultaneously – the InstallHooks will start system event capture and the Activity Recorder will start screen capture. Both applications will be minimzed on your taskbar. The Activity Recorder will also have an icon in the icon area of your taskbar.
4. Your on-screen activities and system events are now being recorded.

To pause recording:

1. To pause the Activity Recorder (the screen capture software) right click the icon and pause from the floating menu. Or you can access the minimized application and directly pause.
2. To pause the system event recording, you should just stop the application and restart when you are ready to record again. Follow the instructions in the next step on how to stop the application. Follow the instructions in the previous step to restart the application.

To stop recording:

1. First exit the Activity Recorder by right clicking on the taskbar icon. You must do this step first before closing the InstallHooks function.

2. After exiting the Activity Recorder, then maximize the Installhooks application. Click on the close button first, then close the application, otherwise the data won't be saved.

Note: Please DO NOT rightclick on the minimized task bar applications (not the icon) and close directly. No data will be saved then.

Recruitment script for Evaluation of Oasis (Chapter 8):

We are looking for users with experience in Microsoft Visio or Visual Studio to perform in a human-computer interaction experiment. The experiment will take place between August 14 and August 28 in the HCI lab and is around 2.5 hours long. You will receive **$50** for participating. An additional $50 award will also be given to the person who is able to produce the best outcome in the study.

If you are interested in participating, please email Shamsi Iqbal at siqbal@uiuc.edu for further information and to set up a time. Available time slots are at: http://www.google.com/calendar/embed?src=stiqbal%40gmail.com , but we are flexible with the slots and will work with you to accommodate your schedule.

This study has been approved by the IRB under protocol #07724.

Thank you,

Brian Bailey and Shamsi Iqbal

Instructions for Evaluation of Oasis (Chapter 8):

Visio Task

For this task, you will design floor plans for several spaces that will be utilized by graduate students in a new computer science building. **There will be a $50 award for the participant who produces the most creative floor plan design.** Please do not be concerned about construction costs or equipment availability; rather, focus on producing the most creative design possible. To increase the likelihood of producing the most creative design, you should generate as many design alternatives as possible during the two hour period. We will consider all of the designs that you generated when selecting the winner.

If you become stuck or just need a short break, please feel free to check e-mail, browse the Web, chat with friends, etc. This is not a test, and we want you to work on this task in a manner similar to how you would work on any interactive task at home or the office.

During the task, you will receive periodic notifications, appearing as pop-ups on the bottom right corner of your screen. Notifications will contain scents to information that may be of interest to you and you have to click on the pop-up to get to a webpage that contains that information. Please try to click on the pop-ups as often as possible. You will get a dialog box asking about the frustration caused by the notification which you have to respond to before getting to the actual page with the notification content. If you fail to respond to the notification pop-up, it disappears after 7 seconds and the information is no longer available.

Design constraints

Your floor plan(s) must include the following:

- Physical work areas to accommodate **six** graduate students.
- A lab space that can be used for group meetings, informal discussions, and user experiments.
- A service room that can be used for preparing meals, making coffee, or just hanging out.

Your floor plan(s) must also balance the following social constraints:

- The spaces must be designed to promote community building, exchanges of ideas, and informal collaboration; and it must feel comfortable and inviting to the students.
- The spaces must enable students to focus on their individual work and minimize overall noise levels.

The creativity of your designs will be judged based on how well they address these constraints.

Procedure

After the experimenter has set everything up, please open Visio. From the right panel, select the

document named **VisioTask.vsd**. This is the document you'll be working on. Please place each design alternative onto a separate "page" within this document.

Shape Templates

We have already made available several templates containing shapes that may be useful in your design. These templates are available from the left panel in Visio. You can access additional shapes by going to File -> Shapes and then choose any template that you may find useful.

Visual Studio

In this task, you will implement a user interface for applying a variety of filters to digital images. The programming code must be written using C# inVisual Studio. Your interface should allow users to open an image, apply filters to the image selected from a drop-down menu, and save the image.

There will be a $50 award for the person who implements the best overall system. Your final solution will be judged based on the number of correctly implemented filters, the quality of the interface, and the efficiency and readability of the underlying source code.

If you become stuck or just need a short break, please feel free to check e-mail, browse the Web, chat with friends, etc. This is not a test, and we want you to work on this task in a manner similar to how you would work on any interactive task at home or the office.

During the task, you will receive periodic notifications, appearing as pop-ups on the bottom right corner of your screen. Notifications will contain scents to information that may be of interest to you and you have to click on the pop-up to get to a webpage that contains that information. Please try to click on the pop-ups as often as possible. You will get a dialog box asking about the frustration caused by the notification which you have to respond to before getting to the actual page with the notification content. If you fail to respond to the notification pop-up, it disappears after 7 seconds and the information is no longer available.

Filters to implement

Your interface should support the following convolution filters:

- Smoothing
- Gaussian Blur
- Sharpen

We have provided an incomplete project for you to work on, which includes the matrices for each of the filters above. We have also provided comments within the code describing each filter in detail to help you

in your implementation. Finally, we have provided an empty windows form that you can build your interface on.

Procedure

After the experimenter has set everything up, please open Visual Studio. From the Start Page, select the project named **VisualStudioTask**. This is the document you'll be working on. Please add resources (e.g. windows forms, code files, etc.) to the project as you find necessary.

Help

You may search the web for resources at any point. However, note that while exact implementations are available online, your score will reduce if your code is copied directly from an existing resource. While we encourage reusing code, we prefer that you be as creative as possible with your own solutions.

You will find two jpg files – flowers.jpg and test.jpg which you can use to test your code.

Post experiment interview for Evaluation of Oasis (Chapter 8):

General questions

1. Did you find certain notifications more disruptive than others during the task? If so, why?

2. Do you receive similar types of notifications while performing your day-to-day work?

3. What would you do to minimize the disruption experienced from those notifications?

(Debrief user about the study goals)

4. Do you think this type of scheduling at certain moments during task execution would help alleviate disruption caused by notifications? Why or why not?

5. Based on your experience in this study, would you be willing to use a fully implemented system to automatically schedule notifications at less disruptive moments?

6. What do you think would be some of the strengths and weaknesses of this approach?

7. Assuming that notifications were to be scheduled, would you prefer receiving pending notifications all at once when you reach the next break in your task, or would you want to receive them one-by-one over a series of breaks?

Per notification questions (video will be available for this step)

Take user through each notification. For each, ask the following questions:

1. Did this notification appear during a transition between two units of action during your task?

2. If so, can you specify whether the transition was between Coarse, Medium or Fine units of action?

3. If you did not feel that this was a transition, can you navigate the video and identify the closest moment that you would have preferred this notification to occur.

4. Can you explain why this moment was preferred?

Bibliography

Adamczyk, P. D. and B. P. Bailey (2004). If Not Now When? The Effects of Interruptions at Different Moments Within Task Execution. Proceedings of the ACM Conference on Human Factors in Computing Systems.

Altheide, D. L. (1996). Qualitative Media Analysis. Newbury Park, CA, Sage.

Altmann, E. M. and J. G. Trafton (2002). "Memory for Goals: An Activation-based Model." Cognitive Science **26**(1): 39-83.

Altmann, E. M. and J. G. Trafton (2004). Task Interruption: Resumption Lag and the Role of Cues. 26th annual conference of the Cognitive Science Society.

Baeza-Yates, R. A. and B. Ribeiro-Neto (1999). Modern Information Retrieval. Boston, MA, Addison-Wesley.

Bailey, B. P., P. D. Adamczyk, et al. (2005). "A Framework for Specifying and Monitoring User Tasks." Journal of Computers in Human Behavior, special issue on attention aware systems: (July/August).

Bailey, B. P., P. D. Adamczyk, et al. (2006). "A Framework for Specifying and Monitoring User Tasks." Journal of Computers in Human Behavior **22**(4): 685-708.

Bailey, B. P. and S. T. Iqbal (2008). "Understanding Changes in Mental Workload During Execution of Goal-directed Tasks and Its Application for Interruption Management." ACM Transactions on Computer Human Interaction (TOCHI) **14**(4): 1-28.

Bailey, B. P. and J. A. Konstan (2005). "On the Need for Attention Aware Systems: Measuring Effects of Interruption on Task Performance, Error Rate, and Affective State." Journal of Computers in Human Behavior, special issue on attention aware systems **(July/August)**.

Bailey, B. P. and J. A. Konstan (2006). "On the Need for Attention Aware Systems: Measuring Effects of Interruption on Task Performance, Error Rate, and Affective State." Journal of Computers in Human Behavior **22**(4): 709-732.

Bailey, B. P., J. A. Konstan, et al. (2001). The Effects of Interruptions on Task Performance, Annoyance, and Anxiety in the User Interface. Proceedings of the IFIP TC.13 International Conference on Human-Computer Interaction, Tokyo, Japan.

Barker, R. G. and H. F. Wright (1954). Midwest and its children: The psychological ecology of an American town. Evanston, IL, Row, Peterson.

Barnes, V. E. and W. P. Monan (1990). Cockpit Distractions: Precursors to Emergencies. Proceedings of the Human Factors Society 34th Annual Meeting, Santa Monica, CA.

Beatty, J. (1982). "Task-evoked Pupillary Responses, Processing Load, and the Structure of Processing Resources." Psychological Bulletin **91**(2): 276-292.

Begole, J. B., N. E. Matsakis, et al. (2004). Lilsys: Sensing Unavailability. Proceedings of the ACM Conference on Computer Supported Cooperative Work, Chicago, Illinois, USA, ACM Press.

Bradshaw, J. L. (1967). "Pupil Size as a Measure of Arousal during Information Processing." Nature **216**: 515-516.

Card, S., T. Moran, et al. (1983). The Psychology of Human-computer Interaction. Hillsdale, Lawrence Erlbaum Associates.

Chong, J. and R. Siino (2006). Interruptions on software teams: a comparison of paired and solo programmers. Proceedings of the 2006 20th anniversary conference on Computer supported cooperative work. Banff, Alberta, Canada, ACM**:** 29-38.

Chung, P. H. and M. D. Byrne (2004). Visual Cues to Reduce Errors in a Routine Procedural Task. Proceedings of the 26th annual conference of the Cognitive Science Society.

Cohen, S. (1978). Environmental load and the allocation of attention. Hillsdale, NJ, Lawrence Erlbaum.

Cohen, S. (1980). "After effects of stress on human performance and socail behavior: A review of research and theory." Psychological Bulletin **88**: 82-108.

Cutrell, E., M. Czerwinski, et al. (2001). Notification, Disruption and Memory: Effects of Messaging Interruptions on Memory and Performance. Proceedings of the IFIP TC.13 International Conference on Human-Computer Interaction, Tokyo, Japan.

Czerwinski, M., E. Cutrell, et al. (2000). Instant Messaging and Interruption: Influence of Task Type on Performance. Annual Conference of the Human Factors and Ergonomics Society of Australia (OZCHI), Sydney, Australia.

Czerwinski, M., E. Cutrell, et al. (2000). Instant Messaging: Effects of Relevance and Timing. People and Computers XIV: Proceedings of HCI, British Computer Society.

Czerwinski, M. and E. Horvitz (2002). An Investigation of Memory for Daily Computing Events. Proceedings of HCI 2002: Sixteenth British HCI Group Annual Conference, London, England.

Czerwinski, M., E. Horvitz, et al. (2004). A diary study of task switching and interruptions. Proceedings of the ACM Conference on Human Factors in Computing Systems.

Dabbish, L. and R. E. Kraut (2004). Controlling interruptions: awareness displays and social motivation for coordination. Proceedings of the ACM Conference on Computer Supported Cooperative Work.

Daniels, J. (2000). Integrating a spoken language system with agents for operational information access. Innovative Applications of Artificial Intelligence (IAAI-2000), Austin, TX.

Degani, A. and E. Wiener (1993). "Cockpit checklists: Concepts, design, and use." Human Factors **35**(2): 345-359.

DeMarco, T. and T. Lister (1999). Peopleware: Productive Projects and Teams 2nd Ed. New York, Dorset House Publishing Company.

Dey, A. K. and G. D. Abowd (2000). CybreMinder: A Context-Aware System for Supporting Reminders. Proceedings of 2nd International Symposium on Handheld and Ubiquitous Computing.

Dismukes, K., G. Young, et al. (1998). "Cockpit Interruptions and Distractions." ASRS Directline **10**.

Dragunov, A. N., T. G. Dietterich, et al. (2005). TaskTracer: a desktop environment to support multi-tasking knowledge workers. Proceedings of the International Conference on Intelligent User Interfaces, San Diego, California, USA.

Fogarty, J. and S. E. Hudson (2007). Toolkit support for developing and deploying sensor-based statistical models of human situations Proceedings of the SIGCHI conference on Human factors in computing systems San Jose, California, USA ACM Press**:** 135-144

Fogarty, J., S. E. Hudson, et al. (2004). Examining the Robustness of Sensor-Based Statistical Models of Human Interruptibility. Proceedings of the ACM Conference on Human Factors in Computing Systems, Vienna, Austria.

Fogarty, J., A. J. Ko, et al. (2005). Examining task engagement in sensor-based statistical models of human interruptibility. Proceeding of the ACM Conference on Human Factors in Computing Systems.

Fogarty, J., J. Lai, et al. (2004). "Presence versus Availability: The Design and Evaluation of a Context-Aware Communication Client." International Journal of Human-Computer Studies **61**(3): 299-317.

Franke, J. L., J. J. Daniels, et al. (2002). Recovering context after interruption. Proceedings of 24th Annual Meeting of the Cognitive Science Society (CogSci 2002).

Fry, E. (1968). "A readability formula that saves time." Journal of Reading **11**(7): 265-271.

Gillie, T. and D. Broadbent (1989). "What Makes Interruptions Disruptive? A Study of Length, Similarity, and Complexity." Psychological Research **50**: 243-250.

Gluck, J., A. Bunt, et al. (2007). Matching attentional draw with utility in interruption Proceedings of the SIGCHI conference on Human factors in Computing Systems San Jose, California, USA ACM Press**:** 41-50

Gonzalez, V. M. and G. Mark (2004). "Constant, Constant, Multi-tasking Craziness": Managing Multiple Working Spheres. Proceedings of the ACM Conference on Human Factors in Computing Systems, Vienna, Austria.

Green, P. (2000). Crashes Induced by Driver Information Systems and What Can be Done to Reduce Them. Convergence 2000, Warrendale, Society of Automotive Engineers.

Guerlain, S. and R. Willis (2001). The tactical tomahawk weapons control system: operator interface design project. Human Systems Integration Symposium.

Hall, M. A. (2000). Correlation-based Feature Selection for Discrete and Numeric Class Machine Learning. Proceedings of the 17th International Conference on Machine Learning, San Francisco, CA.

Hanson, C. and W. Hirst (1989). "On the representation of events: a study of orientation, recall, and recognition." Journal of Experimental Psychology: General **118**(2): 136-147.

Hart, S. G. and L. E. Staveland (1988). Development of a Muli-dimensional Workload Rating Scale: Results of Empirical and Theoretical Research. Human Mental Workload. P. A. Hancock and N. Meshkati. Amsterdam, The Netherlands, Elsevier**:** 138-183.

Heider, F. (1958). The psychology of interpersonal relations. New York, Wiley.

Hess, E. H. (1972). Pupillometrics: A method of studying mental, emotional and sensory processes. Handbook of Psychophysiology. N. S. Greenfield and R. A. Sternbach. New York, Holt, Rinehart & Winston**:** 491-531.

Hess, E. H. and J. M. Polt (1964). "Pupil Size in Relation to Mental Activity during Simple Problem Solving." Science **132**: 1190-1192.

Hoecks, B. and W. Levelt (1993). "Pupillary Dilation as a Measure of Attention: A Quantitative System Analysis." Behavior Research Methods, Instruments, & Computers **25**: 16-26.

Horvitz, E. (1999). Principles of Mixed-Initiative User Interfaces. Proceedings of the ACM Conference on Human Factors in Computing Systems**:** 159-166.

Horvitz, E. and J. Apacible (2003). Learning and Reasoning about Interruption. Proceedings of the Fifth ACM International Conference on Multimodal Interfaces**:** 20-27.

Horvitz, E., J. Apacible, et al. (2005). Balancing Awareness and Interruption: Investigation of Notification Deferral Policies. User Modeling 2005, 10th International Conference. L. Ardissono, P. Brna and A. Mitrovic. Edinburgh, Scotland, UK, Springer. **3538:** 433-437.

Horvitz, E., J. Breese, et al. (1998). The Lumiere Project: Bayesian User Modeling for Inferring the Goals and Needs of Software Users. Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence.

Horvitz, E., A. Jacobs, et al. (1999). Attention-Sensitive Alerting. Conference Proceedings on Uncertainty in Artificial Intelligence**:** 305-313.

Horvitz, E., C. M. Kadie, et al. (2003). Models of Attention in Computing and Communications: From Principles to Applications. Communications of the ACM. **46:** 52-59.

Horvitz, E., P. Koch, et al. (2004). BusyBody: creating and fielding personalized models of the cost of interruption. Proceedings of the ACM Conference on Computer Supported Cooperative Work. Chicago, Illinois, USA, ACM Press**:** 507-510.

Horvitz, E., P. Koch, et al. (2002). Coordinate: Probabilistic Forecasting of Presence and Availability. Eighteenth Conference on Uncertainty and Artificial Intelligence. Edmonton, Alberta, Morgan-Kaufmann**:** 224-233.

Horvitz, E., P. Koch, et al. (2002). Coordinate: Probabilistic Forecasting of Presence and Availability. Eighteenth Conference on Uncertainty and Artificial Intelligence, Edmonton, Alberta, Morgan-Kaufmann.

Horvitz, E., P. Koch, et al. (2005). Bayesphone: Precomputation of Context-Sensitive Policies for Inquiry and Action in Mobile Devices. User Modeling, Edinburgh, Scotland.

Hudson, J. M., J. Christensen, et al. (2002). "I'd be overwhelmed, but it's just one more thing to do": availability and interruption in research management. Proceedings of the ACM Conference on Human Factors in Computing Systems, Minneapolis, Minnesota, USA.

Hudson, S. E., J. Fogarty, et al. (2003). Predicting Human Interruptibility with Sensors: A Wizard of Oz Feasibility Study. Proceedings of the ACM Conference on Human Factors in Computing Systems.

Hytintk, J., J. Tommola, et al. (1995). "Pupil Dilation as a Measure of Processing Load in Simultaneous Interpretation and Other Language Tasks." The Quarterly Journal of Experimental Psychology **48A**(3): 598-612.

Iqbal, S. T., P. D. Adamczyk, et al. (2005). Towards an Index of Opportunity: Understanding Changes in Mental Workload during Task Execution. Proceedings of the ACM Conference on Human Factors in Computing Systems, Portland, Oregon, USA.

Iqbal, S. T. and B. P. Bailey (2004). Using Eye Gaze Patterns to Identify User Tasks. Proceedings of the Grace Hopper Celebration of Women in Computing, Chicago, Illinois, USA.

Iqbal, S. T. and B. P. Bailey (2005). Investigating the Effectiveness of Mental Workload as a Predictor of Opportune Moments for Interruption. Proceedings of the ACM Conference on Human Factors in Computing Systems, Portland, Oregon, USA.

Iqbal, S. T. and B. P. Bailey (2006). Leveraging Characteristics of Task Structure to Predict Costs of Interruption. Proceedings of the ACM Conference on Human Factors in Computing Systems, Montreal, Canada.

Iqbal, S. T. and B. P. Bailey (2007). Understanding and Developing Models for Detecting and Differentiating Breakpoints during Interactive Tasks. Proceedings of the ACM Conference on Human Factors in Computing Systems, San Jose, California, USA.

Iqbal, S. T. and B. P. Bailey (2008). Effects of Intelligent Notification Management on Users and Their Tasks. Proceedings of the ACM Conference on Human Factors in Computing Systems, Florence, Italy.

Iqbal, S. T. and E. Horvitz (2007). Conversation Amidst Computing: A Study of Interruptions and Recovery of Task Activity. The 11th International Conference on User Modeling, Corfu, Greece, Springer Berlin / Heidelberg.

Iqbal, S. T. and E. Horvitz (2007). Disruption and Recovery of Computing Tasks: Field Study, Analysis and Directions. Proceedings of the ACM Conference on Human Factors in Computing Systems, San Jose, California, USA.

Jackson, T. W., R. J. Dawson, et al. (2001). "The cost of email interruption." Journal of Systems and Information Technology **5**(1): 81-92.

John, B. E., K. Prevas, et al. (2004). Predictive human performance modeling made easy. Proceedings of the ACM Conference on Human Factors in Computing Systems, Vienna, Austria.

Juris, M. and M. Velden (1977). "The Pupillary Response to Mental Overload." Physiological Psychology **5**(4): 421-424.

Kahneman, D. (1967). "Pupillary Responses in a Pitch-discrimination Task." Perception & Psychophysics **2**: 101-105.

Kahneman, D. (1973). Attention and Effort. Englewood Cliffs, N.J, Prentice-Hall.

Kahneman, D. (1973). Attention and Task Interference. Attention and Effort. New Jersey, Prentice-Hall**:** 178-202.

Kramer, A. F. (1991). Physiological Metrics of Mental Workload: A Review of Recent Progress. Multiple-Task Performance. D. L. Damos. London, Taylor and Francis**:** 279 - 328.

Kreifeldt, J. G. and M. E. McCarthy (1981). Interruption as a Test of the User-computer Interface. Proceedings of the 17th Annual Conference on Manual Control, Jet Propulsion Laboratory, California Institute of Technology, JPL Publication 81-95.

Latorella, K. A. (1996). Investigating Interruptions: An Example From the Flight Deck. Proceedings of the Human Factors and Ergonomics Society 40th Annual Meeting.

Latorella, K. A. (1998). Effects of modality on interrupted flight deck performance: Implications for data link. 42nd Annual Meeting of the Human Factors and Ergonomics Society.

Latorella, K. A. (1999). Investigating Interruptions: Implications for Flightdeck Performance. Washington, National Aviation and Space Administration.

Lee, J. D., J. D. Hoffman, et al. (2004). Collision warning design to mitigate driver distraction. Proceedings of the ACM Conference on Human factors in Computing Systems.

Maes, P. (1994). "Agents that Reduce Work and Information Overload." Communications of the ACM **37**(7): 30-40.

Maglio, P., R. Barrett, et al. (2000). SUITOR: An Attentive Information System. Proceedings of the International Conference on Intelligent User Interfaces.

Maglio, P. and C. S. Campbell (2000). Tradeoffs in Displaying Peripheral Information. Proceedings of the ACM Conference on Human Factors in Computing Systems.

Mark, G., V. M. Gonzalez, et al. (2005). No task left behind?: examining the nature of fragmented work Proceeding of the SIGCHI conference on Human factors in computing systems Portland, Oregon, USA ACM Press**:** 321-330

McCrickard, D. S., R. Catrambone, et al. (2003). "Establishing Tradeoffs that Leverage Attention for Utility: Empirically Evaluating Information Display in Notification Systems." International Journal of Human-Computer Studies **58**(5): 547-582.

McFarlane, D. C. (1999). Coordinating the Interruption of People in Human-computer Interaction. Proceedings of the IFIP TC.13 International Conference on Human-Computer Interaction.

McFarlane, D. C. and K. A. Latorella (2002). "The Scope and Importance of Human Interruption in HCI Design." Human-Computer Interaction **17**(1): 1-61.

Miyata, Y. and D. A. Norman (1986). Psychological Issues in Support of Multiple Activities. User Centered System Design: New Perspectives on Human-Computer Interaction. D. A. Norman and S. W. Draper. Hillsdale, NJ, Lawrence Erlbaum Associates**:** 265-284.

Monk, C. A. (2004). The effect of frequent versus infrequent interruptions on primary task resumption. Human Factors and Ergonomics Society 48th Annual Meeting.

Monk, C. A., D. A. Boehm-Davis, et al. (2002). The Attentional Costs of Interrupting Task Performance at Various Stages. Proceedings of the Human Factors and Ergonomics Society 46th Annual Meeting.

Monk, C. A., D. A. Boehm-Davis, et al. (2004). "Recovering from interruptions: implications for driver distraction research." Human Factors **46**(4): 650-663.

Mutlu, B., A. Krause, et al. (2007). Robust, low-cost, non-intrusive sensing and recognition of seated postures Proceedings of the 20th annual ACM symposium on User interface software and technology. Newport, Rhode Island, USA ACM**:** 149-158.

Nair, R., S. Voida, et al. (2005). Frequency-based Detection of Task Switches. 19th British HCI Group Annual Conference, Edinburgh, Scotland.

Nass, C., J. Steuer, et al. (1994). Computers Are Social Actors. Proceedings of the ACM Conference on Human Factors in Computing Systems.

Navon, D. and D. Gopher (1979). "On the Economy of the Human Processing System: A Model of Multiple Capacity." Psychological Review **86**: 254-255.

Newell, A. and P. S. Rosenbloom (1981). Mechanisms of skill acquisition and the law of practice. Cognitive Skills and their Acquisition. J. R. Anderson. Hillsdale, NJ, Erlbaum**:** 1-55.

Newtson, D. (1973). "Attribution and the Unit of Perception of Ongoing Behavior." Journal of Personality and Social Psychology **28**(1): 28-38.

Newtson, D. and G. Engquist (1976). "The Perceptual Organization of Ongoing Behavior." Journal of Experimental Social Psychology **12**: 436-450.

Newtson, D., G. Enquist, et al. (1977). "The Objective Basis of Behavior Units." Journal of Personality and Social Psychology **35**(12): 847-862.

Norman, D. A. and T. Shallice (1986). Attention to action: Willed and automatic control of behavior. New York, Plenum Press.

O'Conaill, B. and D. Frohlich (1995). Timespace in the workplace: dealing with interruptions. CHI.

Oberg, B. and D. Notkin (1992). "Error reporting with graduated color." IEEE Software **9**(6): 33-38.

Osgood, S. S., K. R. Boff, et al. (1998). Rapid Communication Display Technology Efficiency in a Multi-task Environment. Human Factors Society 32nd Annual Meeting.

Pashler, H. and J. C. Johnston (1998). Attentional Limitations in Dual Task Performance. H. Pashler. Hove, England, UK, Psychology Press/Erbaum (UK) Taylor & Francis**:** 155-189.

Plaue, C., T. Miller, et al. (2004). Is a Picture Worth a Thousand Words? An Evaluation of Information Awareness Displays. Proceedings of Graphics Interface.

Rich, C. and C. L. Sidner (1998). "COLLAGEN: A Collaboration Manager for Software Interface Agents." User Modeling and User-Adapted Interaction **8**(3/4): 315-350.

Ritter, F. E., G. D. Baxter, et al. (2000). "Supporting Cognitive Models as Users." ACM Transactions on Computer-Human Interaction **7**(2): 141-173.

Rizzolatti, G., L. Fadiga, et al. (1996). "Premotor cortex and the recognition of motor actions." Cognitive Brain Research **3**: 131-141.

Rubinstein, J. S., D. E. Meyer, et al. (2001). "Executive Control of Cognitive Processes in Task Switching." Journal of Experimental Psychology: Human Perception and Performance **27**(4): 763-797.

Sellen, A. J., A. Fogg, et al. (2007). Do life-logging technologies support memory for the past?: an experimental study using sensecam Proceedings of the SIGCHI conference on Human factors in computing systems. San Jose, California, USA ACM**:** 81-90.

Solingen, R., E. Berghout, et al. (1998). "Interrupts: Just a Minute Never Is." IEEE Software **15**(September/October): 97-103.

Speier, C., J. S. Valacich, et al. (1999). "The influence of task interruption on individual decision making: An information overload perspective." Decision Sciences **30**(2): 337-360.

Stanton, N., Ed. (1994). Human Factors in Alarm Design. London, Taylor and Francis.

Stasko, J., T. Miller, et al. (2004). Personalized Peripheral Information Awareness through Information Art. Ubiquitous Computing 6th International Conference (UbiComp).

Stumpf, S., V. Rajaram, et al. (2007). Towards Harnessing User Feedback for Machine Learning. Intelligent User Interfaces. Honolulu, Hawaii, USA, ACM**:** 82-91.

Takahashi, K., M. Nakayama, et al. (2000). The Response of Eye-movement and Pupil Size to Audio Instruction while Viewing a Moving Target. Proceedings of the ACM Conference on Eye Tracking Research & Applications.

Tollinger, I., R. L. Lewis, et al. (2005). Supporting efficient development of cognitive models at multiple skill levels: exploring recent advances in constraint-based modeling. Proceedings of the ACM Conference on Human Factors in Computing Systems, Portland, Oregon, USA.

Trafton, J. G., E. M. Altmann, et al. (2003). "Preparing to resume an interrupted task: effects of prospective goal encoding and retrospective rehearsal." International Journal of Human-Computer Studies **58**: 583-603.

Tullio, J., A. K. Dey, et al. (2007). How it works: a field study of non-technical users interacting with an intelligent system Proceedings of the SIGCHI conference on Human factors in Computing Systems San Jose, California, USA ACM Press**:** 31-40.

Van Dantzich, M., D. Robbins, et al. (2002). Scope: Providing Awareness of Multiple Notifications at a Glance. Proceedings of Advanced Visual Interfaces.

Verney, S. P., E. Granholm, et al. (2001). "Pupillary responses and processing resources on the visual backward masking task." Psychophysiology **38**(1): 76-83.

Wickens, C. D. (1980). The structure of attentional resources. Attention and Performance VIII. R. Nickerson. Hillsdale, NJ, Lawrence Erlbaum**:** 239-257.

Wickens, C. D. (1991). Processing resources and attention. Multiple-task performance. D. L. Damos. London, Taylor & Francis**:** 3-34.

Wickens, C. D. (2002). "Multiple Resources and Performance Prediction." Theoretical Issues in Ergonomic Science **3**(2): 159-177.

Wickens, C. D., J. Helleberg, et al. (2001). Pilot Task Management: Testing an Attentional Expected Value Model of Visual Scanning. Savoy, IL, University of Illinois, Aviation Research Lab.

Yeh, M. and C. D. Wickens (1988). "Dissociation of performance and subjective measures of workload." Human Factors **30**: 111-120.

Zacks, J., B. Tversky, et al. (2001). "Perceiving, remembering, and communicating structure in events." Journal of Experimental Psychology: General **130**(1): 29-58.

Zacks, J. M. and B. Tversky (2001). "Event structure in perception and conception." Psychological Bulletin **127**: 3-21.

Zijlstra, F. R. H., R. A. Roe, et al. (1999). "Temporal Factors in Mental Work: Effects of Interrupted Activities." Journal of Occupational and Organizational Psychology **72**: 163-185.

Author's Biography

Shamsi Tamara Iqbal was born in Guildford, United Kingdom, and moved to Bangladesh at an early age where she completed most of her schooling. She graduated from Bangladesh University of Engineering and Technology (BUET) with honors in Computer Science and Engineering in 2001. Shamsi joined the Masters program in the department of Computer Science at the University of Illinois at Urbana-Champaign in Spring 2002 and switched to pursue a Ph.D. in the area of Human-computer Interaction in Spring 2003. She received a Masters degree in Computer Science from the University of Illinois at Urbana-Champaign in December 2004.

Shamsi's dissertation work has focused on developing and evaluating computational systems for intelligently managing notifications in the desktop. Her broader research interests are in attention management, development of models of user activity, physiological measures as evaluation metrics and educational technology. Shamsi's work on interruption management has been featured in the media, e.g. the New York Times and the American Way magazine in 2007. She has served on program committees of several leading conferences in the field of human-computer interaction, including ACM UIST (Poster committee, 2007) and ACM CHI (Notes committee, 2008). Shamsi will be joining the Adaptive Systems and Interaction groups at Microsoft Research, Redmond as a full-time researcher starting August 2008.